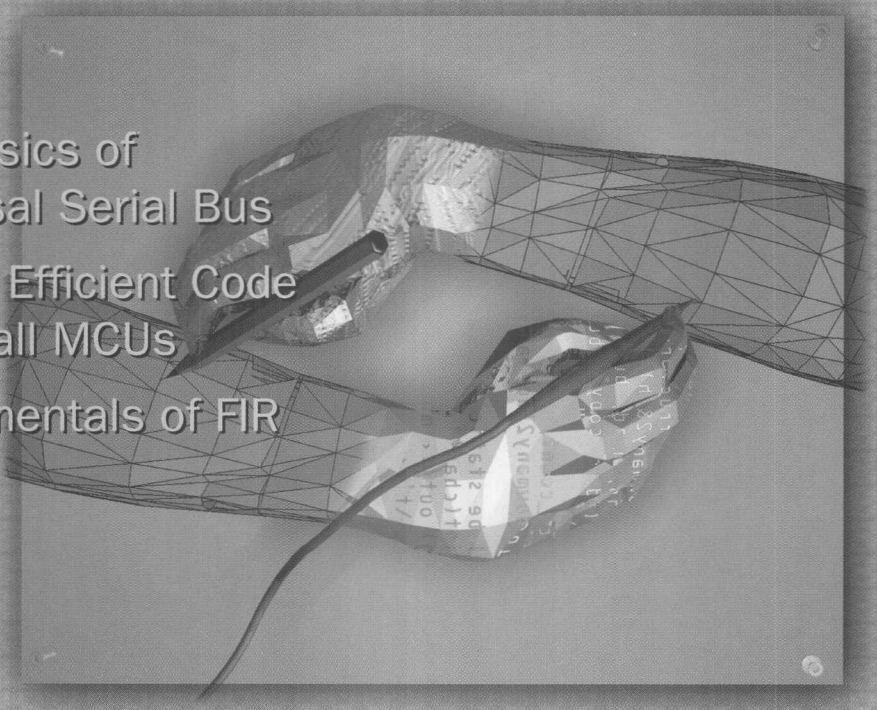
VOLUME 10, NUMBER 6 **U.S. $3.95 CANADA $4.95**
**JUNE 1997** A MILLER FREEMAN PUBLICATION

# Embedded Systems PROGRAMMING

## SPEED TO MARKET WITH HARDWARE-SOFTWARE CO-SIMULATION



The Basics of Universal Serial Bus

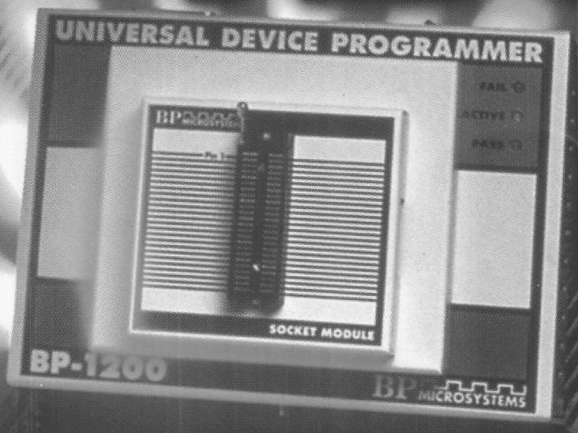
Writing Efficient Code for Small MCUs

Fundamentals of FIR Design

**SPECIAL REPORT:**
**SOFTWARE DEBUGGERS**

www.embedded.com

# BP-1200



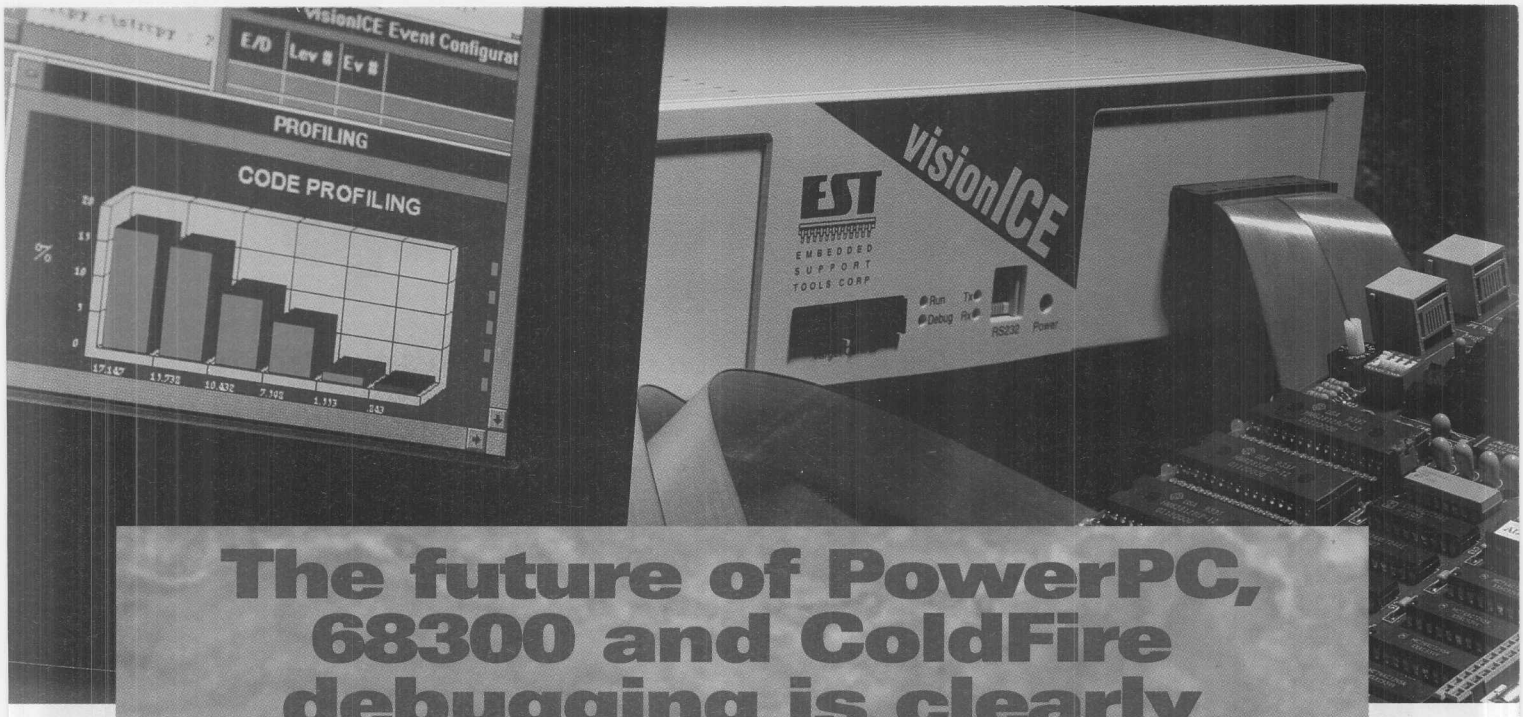

# Proven Performance

The BP-1200 is field upgradeable to 240 pins with full vector and continuity tests. With over 8,500 supported devices, you'll always be able to have the latest technology at your finger tips.

The BP-1200 outperforms its competitors in more ways than one. It's a proven fact. BP Microsystems has built the BP-1200 to program with precision. Only a proven performer like the BP-1200 would offer a three year warranty and toll free technical support. And to top it off, BP Microsystems offers free software updates for the life of the programmer. Prove you're the best by using the best. The BP-1200 from BP Microsystems.

BP MICROSYSTEMS

1000 North Post Oak Road • Houston, Texas 77055-7327 • 713-688-4600 • 1-800-225-2102
Fax: 713-688-0920 • Internet: sales@bpmicro.com • Web Address: http://www.bpmicro.com
CIRCLE # 1 ON READER SERVICE CARD




# The future of PowerPC, 68300 and ColdFire debugging is clearly within vision.

## visionICE: scalable hardware and real-time software debugging for this century and the next.

**PowerPC**

**68360**

**ColdFire**

**6833x**

**68340**

**68341**

**68349**

Your next generation product will be smaller, faster, cheaper and out-the-door in record time. To survive, you'll need a state-of-the-art tool that's affordable for every developer on your team, yet doesn't run out of steam when really nasty bugs surface. You'll need visionICE from EST. Its modular construction delivers the exact amount of power to conquer all your debugging tasks, today and in the future. With visionICE, your tools are never underpowered, never overpriced.

**visionBDM**

Premium BDM emulation and control. It's all you need for rock-solid software debugging.

**visionEVENT**

A full-scale ICE module to expose the toughest real-time bugs. It instantly adds 64K x 192 bit trace, 48 bit time stamp and 64 hardware breakpoints on ranges, externals and mask values.

**visionNET**

Remote, shared debugging and high-speed download from anywhere on your LAN.

**visionMEM/visionFLASH**

4 Mbytes of high-speed overlay memory and up to 32 Mbytes of NV memory for FLASH programming and production test.

**3rd Party Software**

Fully compatible with PassKey, pSOSystem, SingleStep, Tornado/VxWorks and XRAY/VRTX.

EST

**800-957-5588** (in USA) **www.estc.com**

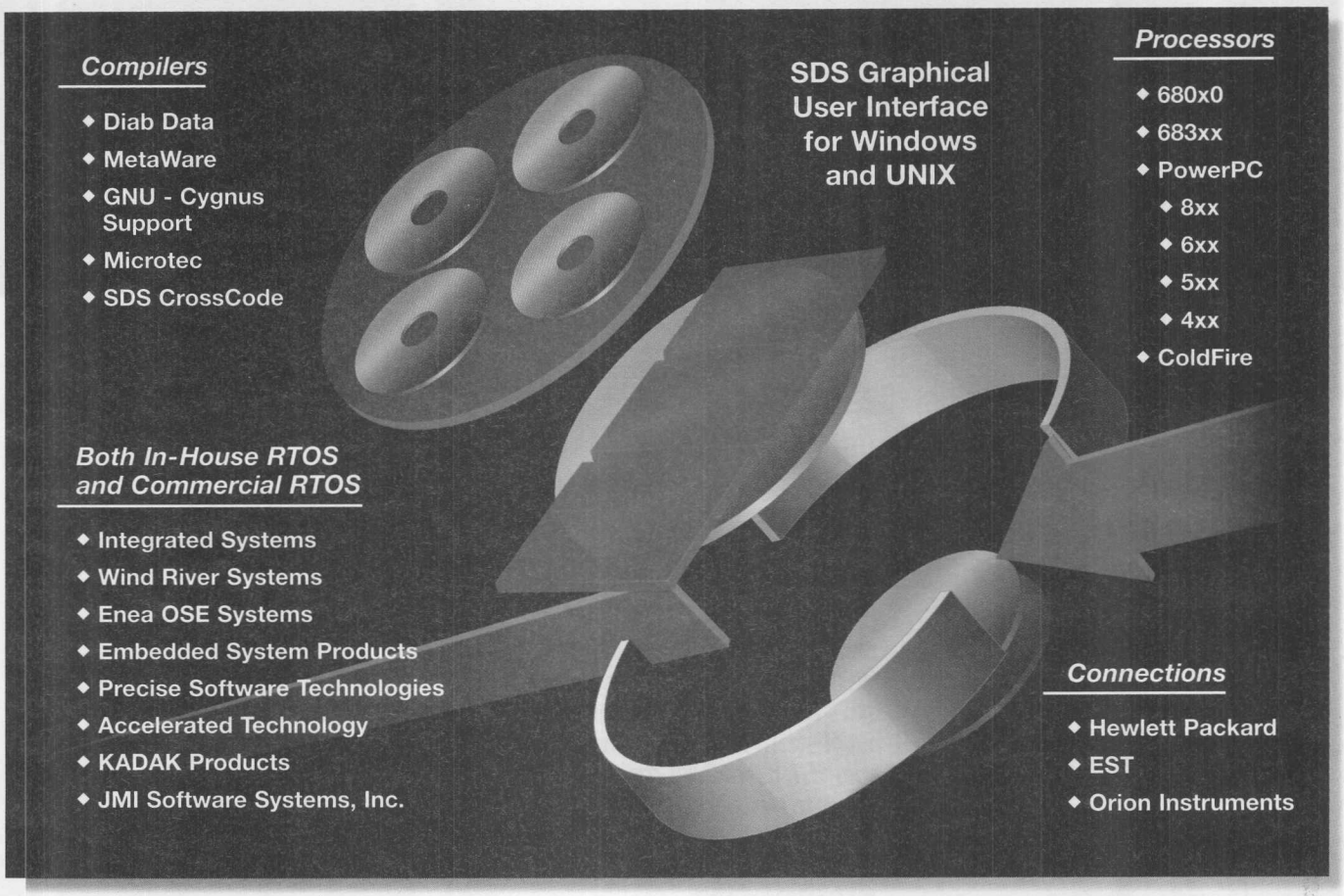
Embedded Support Tools Corp., 120 Royall St., Canton, MA 02021-9725, Tel.: 617-828-5588, Fax: 617-821-2268, European Tel.: 33-130-573200

CIRCLE # 2 ON READER SERVICE CARD

Open Intelligent

# SingleStep™ Debug Suite

*Simulation • On-Chip Debug • Emulation • Target Monitors and Servers*




## PICK THE BEST, SINGLESTEP DOES THE REST.

**WORLD-CLASS • OPEN ENVIRONMENT • FOR THE ENTIRE LIFE CYCLE**

Now you can build your ideal toolset configuration for embedded development. From the components you select. From the vendors you choose. Pick your ideal processor. The fastest compiler. Your favorite real-time kernel. In-house or proprietary. The best debugging tools. Assembled and working together with the latest hardware tools. Integrated with the SingleStep Debug Suite with a common GUI. The SingleStep Debug Toolset is the fastest way to get your next embedded design project up and running—and its open, modular design allows you to keep pace with the latest tools and most advanced processors as they become available. Visit us on the Internet @ www.SDSI.com.

sds

*software development systems*

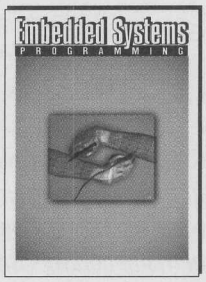
The Most Responsive Software Company

**815 COMMERCE DRIVE, SUITE 250, OAK BROOK, IL 60521 PHONE 630.368.0400, FAX 630.990.4641.**



**CIRCLE # 3 ON READER SERVICE CARD**

# Table of Contents




**ON THE COVER:**

Draw on hardware-software co-simulation to speed your product to market.

Cover illustration by Rupert Adley.

## FEATURES



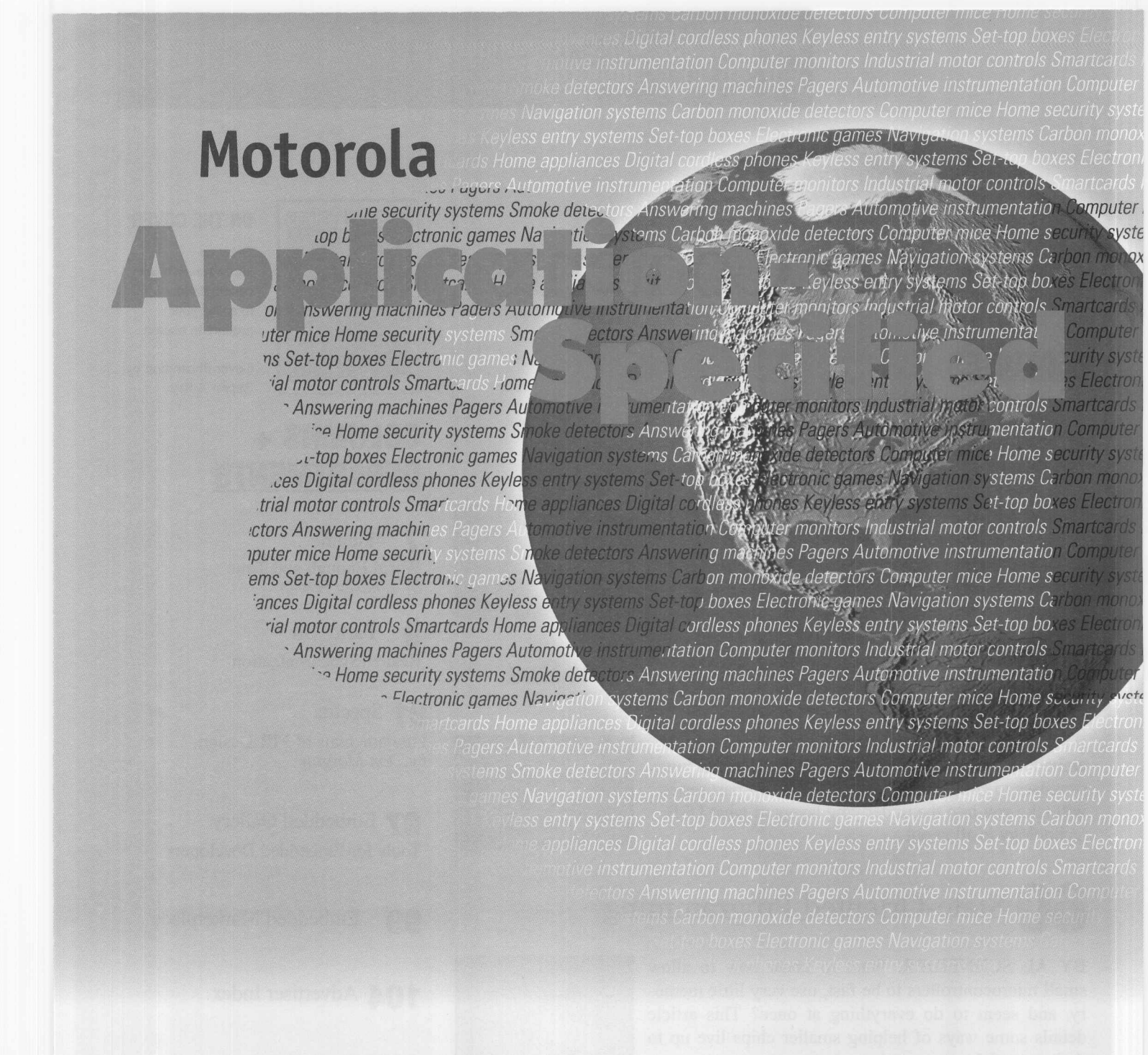
## COLUMNS + DEPARTMENTS



EMBEDDED SYSTEMS PROGRAMMING (ISSN 1040-3272) is published monthly, with an additional issue published in September, by Miller Freeman Inc., 600 Harrison St., San Francisco, CA 94107, (415) 905-2200. Please direct advertising and editorial inquiries to this address. SUBSCRIPTION RATE for the United States is $55 for 13 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of $6 per year. All other foreign subscriptions must be prepaid in U.S. funds with additional postage of $15 per year for surface mail and $40 per year for airmail. POSTMASTER: All subscription orders, inquiries, and address changes should be sent to EMBEDDED SYSTEMS PROGRAMMING, P.O. Box 420046, Palm Coast, FL 32142-0046. For customer service, telephone toll-free (800) 829-5537. Please allow four to six weeks for change of address to take effect. Periodicals postage is paid at San Francisco, CA and additional mailing offices. EMBEDDED SYSTEMS PROGRAMMING is a registered trademark owned by the parent company, Miller Freeman Inc.  EMBEDDED SYSTEMS PROGRAMMING is available on microfilm/fiche from University Microfilms International, 300 N. Zeeb Rd., Ann Arbor, MI 48106, (313) 761-4700.

# Motorola

# Application Specified



Over two billion 68HC05 microcontrollers are now in operation around the world. Powering an ever-growing range of consumer, industrial, computing, communications, and automotive products.

Our 68HC05, the world's most popular MCU, is "customer-specified" for your system. In addition to the 150 (and growing) types of 68HC05 MCUs, you can now select one of our fully upward object code compatible 68HC08 microcontrollers.

The 68HC08 Family offers even higher-performance with faster clock speed options and more features for your system design. But the 68HC08 doesn't just provide more instructions and registers, it also runs 68HC05 object code.


# The world's first choice.

And both the 68HC05 and 68HC08 Families offer complete support and a full suite of third-party development tools.

So, if your system needs a "world-class" 8-bit solution, specify the world's first choice: Motorola. Our global manufacturing is already ramped-up to deliver production volumes anytime, anywhere.

For more information on either the 68HC05 or 68HC08 Families, call **1-800-765-7795** ext. 882 or request by FAX at **1-800-765-9753**.

You may also visit our Web site at **http://sps.motorola.com/csic**. When it comes to 8-bit solutions, there's a world of difference with Motorola MCUs.

MOTOROLA

CSIC Microcontroller Divisions

*What you never thought possible.*™

Visit us on the World Wide Web

http://www.keil.com/

KEIL SOFTWARE

# Embedded Software Development Tools

*For the Siemens 166, C167, C165, and C163 microcontroller family and the NEW! C161*

## C166 Version 3.0 — The High-Performance C Compiler for the Siemens 166 Family

Keil Development Tools unlock the features and performance of the Siemens 166 and C167 and make you an instant 16-bit embedded expert. The **µVision Windows-based IDE** encapsulates your project with complete compiler and linker controls and helps you create complex programs in record time! The dScope symbolic, source-level debugger lets you simulate and debug your target system including external hardware.

The ANSI-standard **C166 compiler** is designed specifically for the 166 and C167 families—language extensions give you access to all CPU resources including the PEC, interrupts, SFRs, and DPPs.

C166 is the most efficient, flexible development tool set available today. Support for all derivatives and compatibility with the **major emulator vendors** makes C166 the best choice for your 166 and C167 projects!

**PK166/PK161 Highlights**

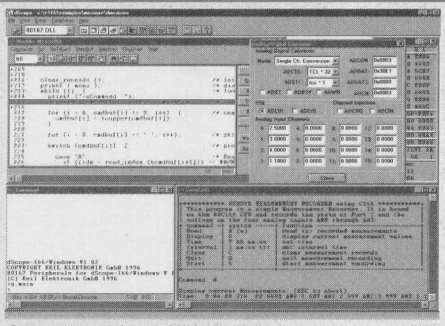
- ANSI-compliant C Compiler with C Libraries
- Includes Memory Allocation Routines
- Floating-Point Libraries Included
- Support for Structures and Unions
- Interrupt Functions may be written in C
- Reentrant Functions
- Parameters Passed in Registers
- C Support for all Special Function Registers
- Supports the Entire 16M Address Space
- Inline Assembly Code Support
- Includes Macro Assembler
- Includes Single-Chip Real-Time Operating System
- Windows-based IDE and Debugger
- Free Updates via the World Wide Web
- Free Update Notification via E-mail
- Free One-Year Technical Support

## dScope Source-Level, Symbolic Debugger and Target Monitor

The **dScope source-level, symbolic simulator/debugger** helps you test and debug your 166 and C167 application programs. dScope provides:

- Execution, conditional, and memory access breakpoints,
- Watchpoints for all variable types,
- Mixed source/assembly display,
- Software performance analysis,
- Code coverage analysis,
- User and signal functions,
- and On-chip peripheral support.

Debugging your target hardware is easy when you use the **MON166 monitor** and dScope debugger. MON166 is a full-featured, royalty-free target monitor designed for the 166 and C167 families. It can be configured for a wide variety of systems—even those with bootloader capabilities. Using dScope and MON166, you can easily view program source code, watch special variables, and **examine target memory**! And, MON166 comes preconfigured for a variety of third-party evaluation boards.



dScope provides you with a powerful debugging platform that completely simulates **all** 166 and C167 derivatives including the C165, C163, and C161 devices. Using CPU driver DLLs, dScope gives you dialog box access to all on-chip SFRs. You can easily view and modify SFR contents for A/D converters, timer/counters, ports, and serial ports.

## RTX166 — Real-Time Operating System

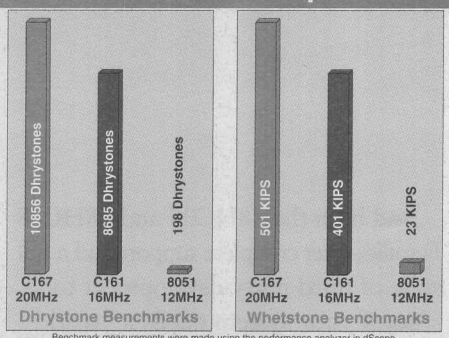
**RTX166** is a multitasking real-time operating system that supports the entire Siemens 166 and C167 family. RTX166 makes designing complex, time-critical software projects easy by providing sophisticated management for multiple tasks running on a single CPU.

**RTX166 Tiny** supports single-chip applications where code and memory space must remain at a minimum. RTX166 Tiny lets you create and delete tasks and send and receive signals.

**RTX166 Full** supports applications where robust features are required. In addition to the features found in the tiny RTX, the RTX166 Full version manages interrupts, resources (via semaphores), and memory pools. Mailbox, system clock, and task management routines are also included.

RTX166 Full provides support for the **C167CR CAN interface**. This lets you get started with the CAN interface as quickly as possible.

## Performance Comparison




Benchmark measurements were made using the performance analyzer in dScope. C167 and C161 programs were compiled for 16-bit, non-multiplexed bus mode with 0 wait-states.

## Call 1-800-521-4957 for a FREE Evaluation CD-ROM of all our tools!


**United States:**
**Keil Software, Inc.**
16990 Dallas Parkway, #120
Dallas, Texas 75248-1903
USA

Phone 972-735-8052
Fax 972-735-8055
BBS 972-713-9883
E-mail sales.us@keil.com
support.us@keil.com

**Europe:**
**Keil Elektronik GmbH**
Bretonischer Ring 15
85630 Grasbrunn
Germany

Phone ++49 89 / 456040-0
Fax ++49 89 / 468162
BBS ++49 89 / 4606286
E-mail sales.intl@keil.com
support.intl@keil.com


US Distributors: **Ceibo:** 314-830-4084, **CMX:** 508-872-7675, **Emulation Technology:** 408-982-0660, **HiTools (Hitex):** 800-454-4839, **Metalink:** 602-926-0797, **Micom Systems:** 214-245-2533, **Microware Technology:** 619-693-4280, **Nohau:** 408-866-1820, **Peachtree Technology:** 770-888-4002, **Signum Systems:** 805-371-4608.

International Distributors: **AUS:** Electro Optics 02/9654 1873, **A:** Rekirsch 01/259 72 70 0, **B:** Bytecom 010/22 34 55, **Brazil:** Anacom 11/453 5588, **Canada:** Ximetrix 905-602-8550, **Czech:** ComAp 2/6833 858, EDI 2/02 2683, **CH:** Thau 01/745 18 18, **DK:** Nohau 043/44 60 10, **E:** CAPEL 93/291 76 33/34, **F:** Antycip 1/3961 1414, **Hong Kong:** Oritental 852/2402 3200, **I:** Microtask 02/4982051, Nord 02/66092.1, **India:** Embedded Systems Solutions 80/3323029, **IRL:** Ashling 061/33 44 66, **Israel:** ITEC 03/6 49 1202, **Japan:** NPS 03/3226 8110, **Korea:** Hankook 02/783 0022, **Malaysia:** Flash 603/7168729, **NL:** Tritec 078/681 6133, **NZ:** Electro Optics 0800/44 2148, **PL:** WG 02/621 77 04, **Romania:** ASTI 3125907, **S:** Nohau 040/59 22 00, **Singapore:** Flash 65/749 6168, Testech 65/7492162, **SLO:** ASYST 061/445526, **S. Africa:** EBE 012/803 7680/93, Kiberlab 012/660 2752, **Taiwan:** Deemax 3/5232548, **Turkey:** EMPA 1/599 30 50, **UK:** Hitex 01203/69 20 66, Nohau 01962/73 31 40.

**CIRCLE # 4 ON READER SERVICE CARD**

Embedded Systems PROGRAMMING

EDITOR-IN-CHIEF
Lindsey Vereen

MANAGING EDITOR
Mariska van Aalst

TECHNICAL EDITOR
Nicholas Cravotta

COPY/PRODUCTION EDITOR
Michael Shapiro

CONTRIBUTING EDITORS
Jack W. Crenshaw
Jack G. Ganssle
Larry Mittag
Don Morgan
P.J. Plauger
Tyler Sperry

PUBLISHING DIRECTOR
Donna J. Esposito

PUBLISHER
Mike Flynn, (415) 278-5251

NATIONAL SALES MANAGER
Eric Berg, (415) 278-5220

CA REGIONAL SALES MANAGER
Thomas Vilms, (415) 278-5223

WESTERN SALES ASSISTANT
Robin Lander, (415) 278-5274

EASTERN REGIONAL SALES MANAGER
Damon Graff, (617) 235-8258

EASTERN SALES ASSISTANT
Ryan Sorley, (617) 235-8255

PRODUCTION COORDINATOR
James Whitehead

GRAPHIC DESIGNER, ELECTRONICS
Katie Symons

GROUP CIRCULATION MANAGER
Sherri Gronli

ASSISTANT CIRCULATION MANAGER
Jennifer Armstrong

SUBSCRIPTION CUSTOMER SERVICE
(800) 829-5537
Non-U.S. subscribers: (904) 445-4662
embedded@palmcoastd.com

REPRINTS
Andrea Ramiza, (714) 376-6273

Visit the *Embedded Systems Programming* Web Site
**www.embedded.com**

Miller Freeman
A United News & Media publication

CHAIRMAN/CEO
Marshall W. Freeman

EXECUTIVE VICE PRESIDENT/COO
Donald A. Pazour

SENIOR VICE PRESIDENT/CFO
Warren "Andy" Ambrose

SENIOR VICE PRESIDENTS
H. Ted Bahr
David Nussbaum
Wini D. Ragus
Darrell Denny
Galen A. Poss
Regina Starr Ridley

VICE PRESIDENT/PRODUCTION
Andrew A. Mickus

VICE PRESIDENT/CIRCULATION
Jerry M. Okabe

SENIOR VICE PRESIDENT
ELECTRONICS/DESIGN DIVISION
H. Ted Bahr

BPA INTERNATIONAL



# World Enough and Time

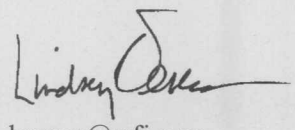
Several years ago, perhaps in the late '70s, a company embarked on the development of a turntable that could play vinyl records with a laser pickup. It had no needle to scratch the delicate vinyl; instead it read the grooves optically. Product development was plagued with glitches and cost overruns. I don't think this novel turntable ever made it to your local audio store, but it was a great idea right up to the very moment that compact disc players hit the market. Thereafter it sank like a stone into the slough of missed marketing opportunities. Herein lies the essence of the time-to-market issue.

Cliché though it may be, you ignore time to market at your peril. According to a marketing guru (is that an oxymoron?), products that get to market six months late but on budget generate about 33% fewer profits over a five year span than they would have had the product shipped on schedule. Marketers have had drilled into them that 60% of a product's return comes within the first six months, which you'll lose if you're late to market. (Of course, this won't stop these same marketers from trying to get you to implement new features near the end of the development cycle, but that's another story.)

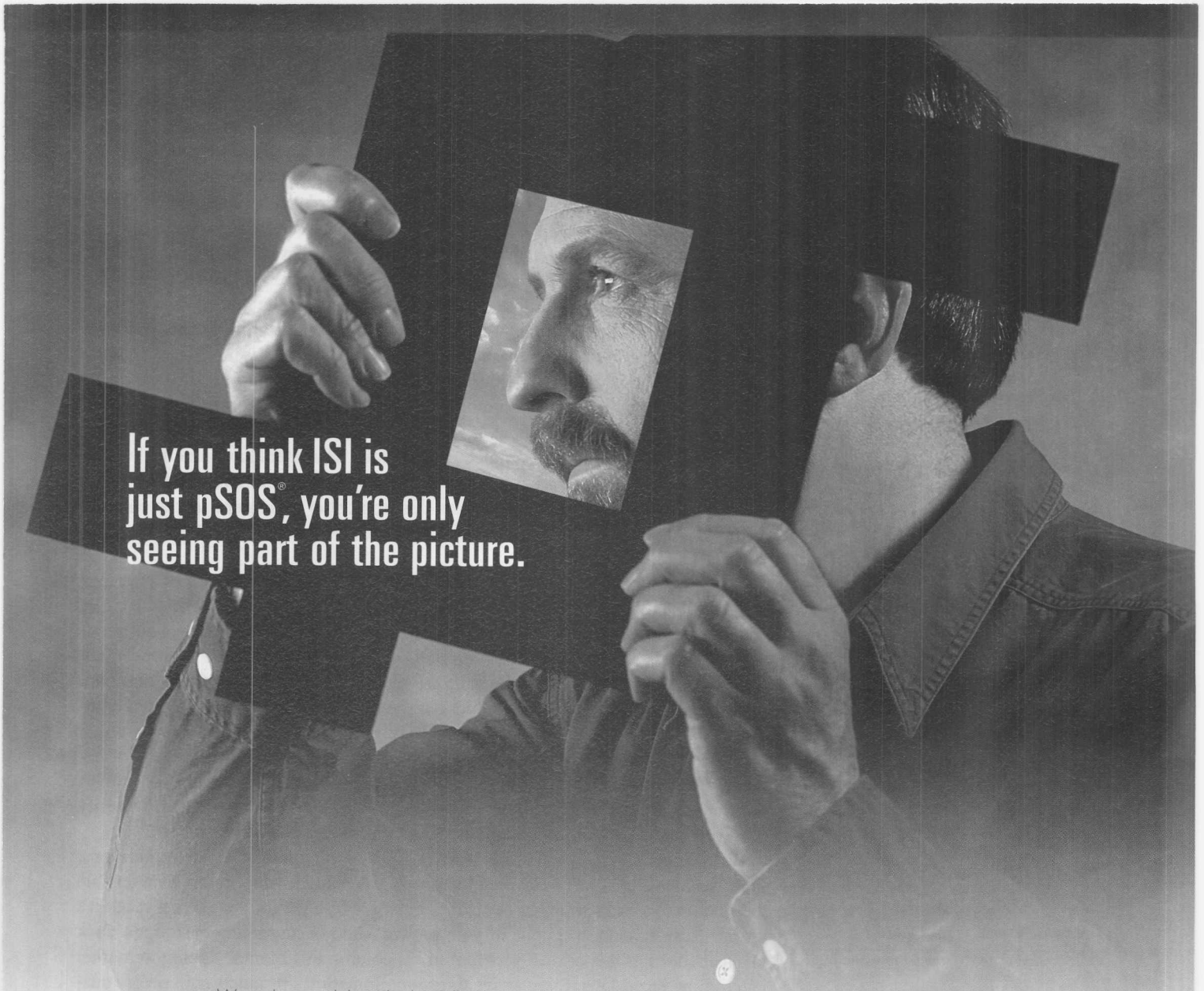
It should be pretty straightforward to calculate the cost of the bill of materials and the related development costs, but once you factor in time to market, the entire equation can change. What if you spend more on your processor but get your product to market six months sooner? Your parts cost is higher but so are your revenues, at least theoretically. Money spent on development tools and methodologies are justified if they help get your product out the door faster. Programming in a high-level language vs. assembly language may produce less efficient code, but it increases productivity. The ability to reuse code can potentially reduce the development time and costs. Hardware/software co-design, the subject of this month's cover story, is emerging as a methodology that can help you to design and verify hardware and software in parallel and to accelerate product development.

But while shipping your product sooner is a worthy goal, there are conditions. First, you have to determine when the product is ready to ship. It can be a terrible temptation to a cash-strapped company to ship before the product is ready, but it's no fair foisting alpha versions of your product onto unsuspecting customers. Jack Ganssle looks at this critical problem in his column this month. Racing to market can also be hazardous if you arrive too soon. Sometimes products ship before customers are ready to accept them. DVD may be an example. Some people fear that HDTV may potentially be another. It's instructive to note that a few companies have gone under, not because they failed to deliver their product, but because they delivered it before the market was ready for it.

You won't run short of ways to spend money on product development. The tricky part is determining which ones will benefit you in the long run. Most frustrating is that you'll never know for sure how you would have fared had you delivered your product six months earlier or six months later. You can only speculate. Of the many potential time-to-market tools, the one that would really help is a time machine.



lvereen@mfi.com




We make your job easier by delivering much more than just pSOSystem™— the industry's leading RTOS. The fact is we provide the greatest breadth and depth of technology, tools and services in the embedded systems industry.

And as new technologies emerge, like Internet and Java,™ you can be sure we're one step ahead. Because we're always looking at the big picture.

Our range of solutions includes:

**RTOS**

pSOSystem™: Reliable, scalable RTOS, proven across the industry's largest installed base.

**Tools**

pRISM+™: Powerful, integrated graphical software development environment.

**Network and Application Components**

Epilogue™: Industry's leading platform-independent protocol software.

pSOSystem Components: Industry-specific application modules.

**Engineering Services**

Dr. Design: Foremost providers of innovative design solutions.

integrated systems

www.isi.com

800.543.7767

Integrated Systems, Inc. 201 Moffett Park Drive, Sunnyvale, CA 94089

pSOS is a registered trademark and pSOSystem, pRISM+ and Epilogue are trademarks of Integrated Systems, Inc. Java is a trademark of Sun Microsystems, Inc.

**CIRCLE # 5 ON READER SERVICE CARD**

# Reader Synchronization

## Policing Asynchronicity

I'd like to make some comments about Bill Lamie's article "Multitasking Mysteries Revealed" (February 1997, p. 38). Not once was the term "asynchronicity" mentioned in the article. Sure, ease of development, portability, and maintainability are important features, but they can all be acheived to a high degree without a multitasking kernel.

As to Lamie's explanation of responsiveness, throughput, and reduced overhead, one might think that the only alternative to using a multitasking kernel is to use polling! As we all know, this isn't the case—interrupts are used (maybe even to a larger extent) in non-multitasking systems. The only compelling reason to use a multitasking kernel is the temporal behavior of a system interacting with its environment, and more specifically the requirement that the system handle asynchronous events. Some systems that need to fulfill this requirement can get by with interrupt handlers and a control loop; however, interrupts have their own priorities, and large pieces of code in an interrupt handler will mask out lower priority interrupts, thus reducing overall performance.

Furthermore, it is the asynchronous nature of a system interacting with its environment that should dictate the division of the software into tasks and, in most cases, will also dictate the relative priorities between those tasks. Finally, only by adopting such a train of thought will developers manage to avoid the pitfalls of multitasking, such as starvation, priority inversion, and excessive overhead. Formal methods such as Temporal Logic offer a good start, but experience has taught me that a deep understanding of the interaction between asynchronous events (and not only the events themselves) will at most times suffice.

**The only reason to use a multitasking kernel is the temporal behavior of a system interacting with its environment.**

G.C. Hillel
ReTIC Systems Ltd.
retic@trendline.co.il

Author Bill Lamie responds: *Upon further review of my article, I wish that I had addressed the proper use interrupts in embedded systems. It is certainly true that asynchronous responsiveness can be addressed solely with interrupts. However, this is typically only possible in small systems that have just one important interrupt. Even in such systems, a large interrupt handling routine might not be feasible if the underlying interrupt frequency is high. Furthermore, most processors have a single interrupt execution level (the Motorola 68K is an exception to the rule), which again restricts the applicability of this technique to small, somewhat single-purpose applications. Finally, tying an application's real-time capabilities exclusively to interrupt handling places the burden of processor allocation back in the application and increases processor dependence. All of this makes the application harder to maintain and to migrate to new processor families.*

*As for not mentioning the term "asynchronicity," I think the concept of trying to guarantee responsiveness to asynchronous external events was sufficiently addressed by my article. I concur that handling asynchronous system behavior is an important reason to use multitasking. However, an equally important feature of multitasking is that it facilitates the division of application software into components or objects. These components or objects are more easily reused, removed, added, and modified. As a result, the division of tasks should generally be based on software components and their priorities should be selected such that they satisfy the system's real-time requirements. As my article mentions, care must be taken regarding how many tasks and priority levels are used.*

*As for Temporal Logic, I don't know how easily this formal method scales to larger, heterogeneous applications. It does seem as though it might be useful for state machines, and perhaps the logic inside of individual software components.*

*As my article pointed out, simple applications may not benefit from multitasking. However, applications that have multiple heterogeneous duties to perform, while at the same time handling asynchronous events in real time, could certainly make good use of a multitasking environment.*

## Madness To His Methods?

Generally speaking, I agree with Orv Balcom's ideas in his article "Control Embedded Projects with Software

Requirements Specs" (February 1997, p. 52), but I do not agree with his methodology.

Balcom points out many good ideas for what should be included in an SRS, but I take exception to a few parts. Balcom's development methodology describes the classic Waterfall development process, which has fallen under great scrutiny because it lacks the early feedback mechanisms that promote rapid product development and high customer acceptance. I'd like to comment on his model:

*1) The customer writes the Project Requirements Specification.* I've been in this business too long to know that customers, be they internal or external, rarely have a clue what they want, let alone how to write it down in a clear, concise, and unambiguous way. I'd be more inclined to have a two-person team, one from the customer and one from the development team, write the document together.

*2) Design the program/write the SRS.*

*3) Review the SRS. If it is wrong, go to Step 2. If the PRS is wrong, go to Step 1.* If the PRS is wrong at this stage, you have an expensive defect because the program is designed and the SRS is written. Perhaps a better Step 2 would have been to carefully review the PRS when it was complete, minimizing the errors coming into the Design/SRS stage.

*4) Code the program.*

*5) Test the program. If code problems return to Step 4, if design problems return to Step 2, if PRS problems return to Step 1.* Now you have really expensive PRS and SRS defects on your hands. You're at final acceptance and the product doesn't meet the customer's expectations.

In my opinion, a spiral development model would be better suited to this type of development. Get the customer to agree upon a core set of requirements. Design and develop that core set of requirements, and show it to the customer early. Get the customer's feedback and specify the next set of deliverables. Of course, there is the risk that the customer will hate this early prototype, but it's better to find

that out after you have spent 25% of the project budget rather than 90% (or worse yet, 150%). This also helps to focus the development team on providing the requirements that the customer needs most, first.

Likewise, using an ASCII editor for spec development is short sighted, as it doesn't allow the use of some of the modern Requirements Management tools which are now commonly available. There is more than the SRS to controlling any project. What if you get to Step 5 and discover that the PRS is wrong? You must now update the PRS, SRS, design, test plans, test procedures, and customer manuals. Do you have any method of assessing impact on those documents? With some of the modern tools, this is easy.

A question that comes to mind is how widely accepted is a 120–page document? Do the developers all have a paper copy or do they access the document electronically? Paper copies are notoriously difficult to keep up-to-date. You specifically define many sections to your SRS; why not make each one of them a separate electronic document? You can link these together to create a master table of contents. I would stay away from paper copies and encourage the developers and customer to access them electronically as much as possible.

Requirements Management makes good business sense for everyone. Minimizing the amount of effort spent on Requirements Management and maximizing the benefit is where the payoff is.

Wayne Woodruff
wayne@bmwmat.bucks.pa.us

Author Orv Balcom responds: *I feel Mr. Woodruff has confused the typical embedded system project with a data processing project. The embedded systems projects with which I have been involved were based on specific requirements, not "expectations" by the customer, who was usually not the end user. These resultant products would often be produced in hundreds or thousands and sometimes with mask programmed ROM. The projects were either fixed-price or had a very constrained budget. The incremental ("spiral") development of the product specification was not an option. In rereading the articles on the Octopus Method (Awad, Kuusela, and Ziegler, September, October, and November, 1996), I find that these authors also do not consider incremental development of the product requirements.*

*This is because any significant change after the project has started is truly a change of scope and will require a modification of the budget and schedule. In most cases, it will also compromise the final product because of the attempts to minimize the impact of the new requirements.*

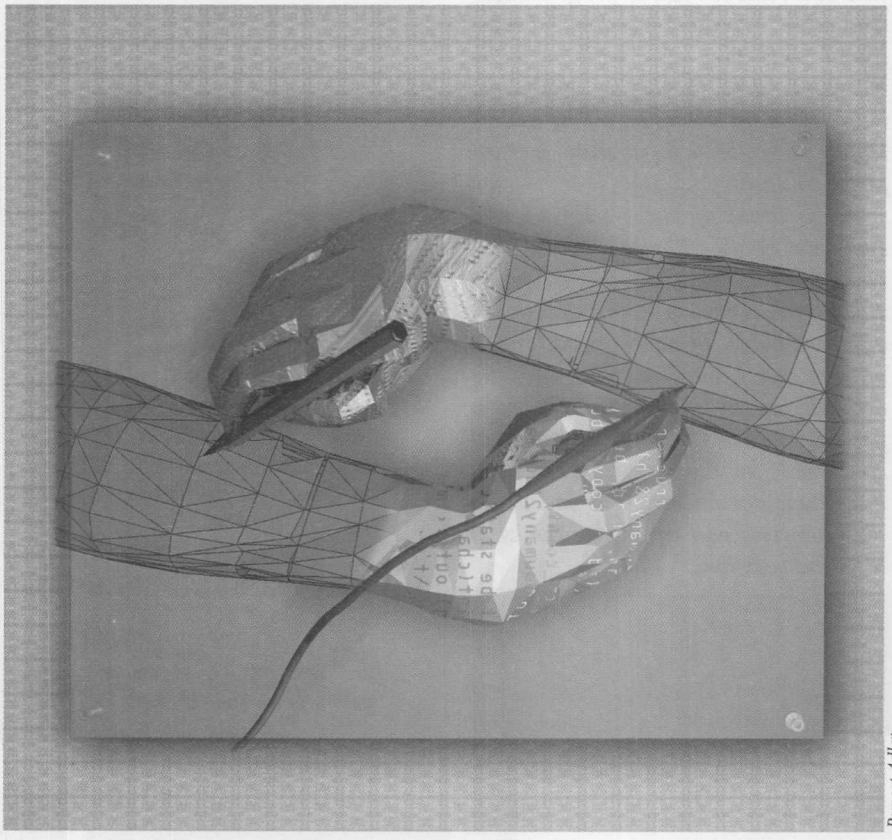
*I am not familiar with the "Requirements Management" tools of which Mr. Woodruff speaks. My hunch is that if they are similar to some of the CASE tools described in the trade press, their cost would exceed the development cost of a typical 8-bit micro project. What I said in my article was that most word processors have proved inadequate. Large documents are not easily accepted by managers who don't like to read or programmers who crave "creative freedom." I have found that they are welcomed by mangers who want a definitive description (in a language they understand) of what the program is supposed to do.*

*As to breaking the document into separate books, I much prefer one book with Post-its on the pages to which I am referring than half a dozen different books at my workstation. In no way can I imagine trying to write code or use an ICE with eight more pop-up windows with a sentence of spec each on my screen. Also, with individual documents, it is inevitable that someone will try to use books of different revisions.*

**We want to know what's on your mind. Please send any questions, criticisms, rants, or raves to Editor-In-Chief Lindsey Vereen at lvereen@mfi.com. We reserve the right to edit all letters for clarity and length.**

# Co-Simulating Software and Hardware in Embedded Systems

Fighting time-to-market issues and wondering if hardware/software co-simulation would help? These two engineers from Sandia Labs review and compare different modeling approaches you can take and then describe how they addressed the problem.



Rupert Adley

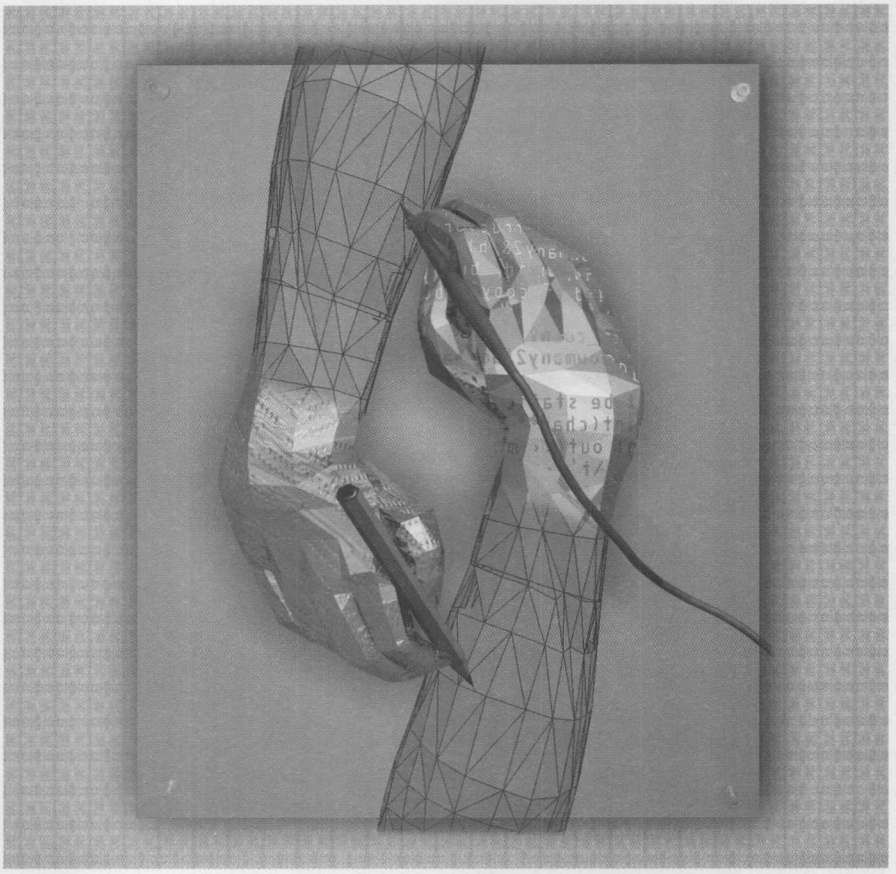
From the time of the first use of microprocessors and microcontrollers in embedded systems, software has been blamed for products being late to market. This stigma results from software being developed after hardware is fabricated. During the past few years, the use of hardware description (or design) languages (HDLs) and digital simulation have advanced to a point where the concurrent development of software and hardware can be contemplated using simulation environments. This alternative offers the potential of 50% or greater reductions in time-to-market for embedded systems.

This article presents a tutorial on the technical issues that underlie hardware-software (hw/sw) co-simulation and the current state of the art. We review the traditional approach to sequential hw/sw design, and suggest a model for concurrent design, which is supported by co-simulation of software and hardware. We follow this discussion with sections on HDLs, modeling and simulation; hardware-assisted approaches to simulation; microprocessor modeling methods; brief descriptions of four commercial products for hw/sw co-simulation; and a description of our own experiments to develop a co-simulation environment.

## TRADITIONAL DEVELOPMENT CYCLE

The workflow in a traditional embedded project is shown in Figure 1. Software is typically developed after a hardware design has begun to stabilize (some actually do!) and prototypes are available for integrating the software and hardware. Add to this the limited observability of the operation of the hardware as the software executes, and the inability to control all of the elements of the

**As per tradition, software still waits for hardware prototypes before significant progress is made on integration.**



design, especially peripheral components running synchronously from the microprocessor or microcontroller, it becomes obvious why the hw/sw integration and test is the most time consuming part of the project. (Almost all engineering veterans have at least one horror story of some heroic effort that involved sleeping under their desk.)

Attempts at concurrent hardware and software development have resulted in only a small portion of software bugs being found early. Software still waits for hardware prototypes before significant progress is made on integration. So, hw/sw integration problems are still found late in development and solved in software. Historically, the nature of technology has made this late discovery a fact of life. However, advances in hardware modeling and simulation offer the potential to develop a new design methodology using hw/sw co-design and co-simulation.

## CO-DESIGN AND CO-SIMULATION

The process of concurrently developing hardware and software encompasses two main areas of study: co-design and co-simulation. In the context of this article, the term *co-design* refers to the process of translating the requirements for a desired system level functional behavior into a partition of hardware and software designs which, taken together, provide and maintain the desired system behavior. The term *co-simulation* is simulation of a model of the hardware running the software, simultaneously providing visibility (and control, to a lesser degree) of the hardware model while allowing software execution to be controlled and observed at all levels of detail necessary for understanding the behavior of the system. The co-simulation environment should provide a user interface that is consistent with the current state-of-the-art for both the hardware simulators and the software emulators used by the development team. In an ideal world, this user interface would provide the same look and feel as the equipment used to test and verify the actual hardware after it has been produced.

**FIGURE 1**
*Traditional embedded systems workflow.*

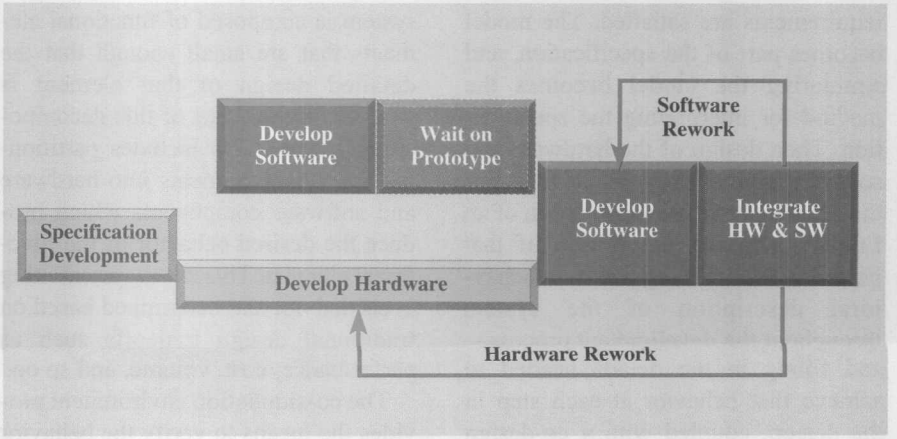
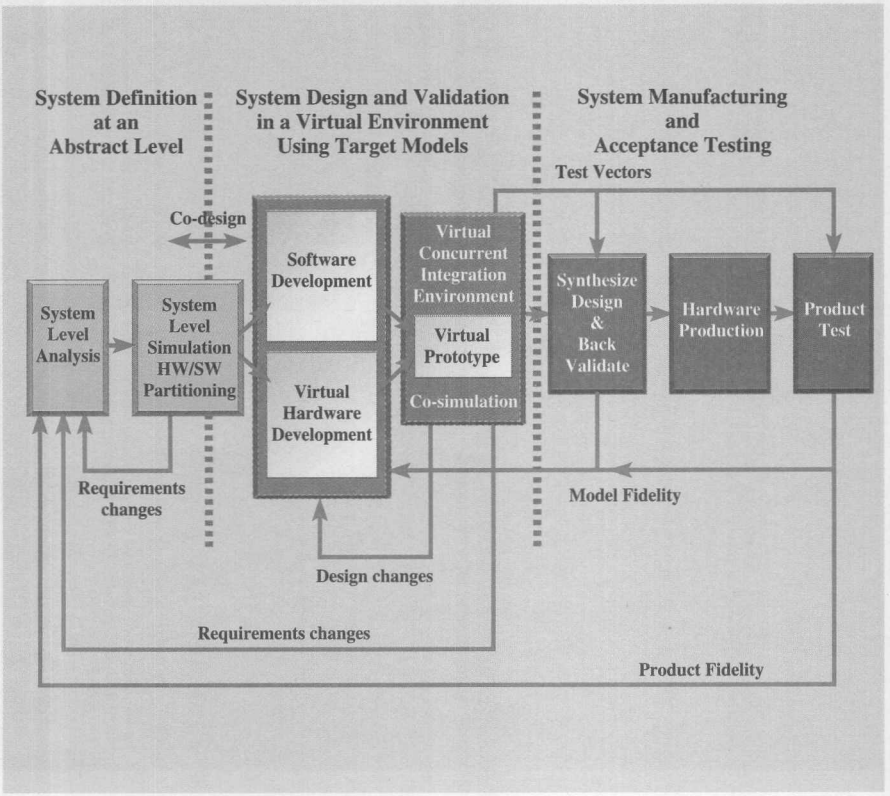


**FIGURE 2**
*New model for concurrent software and hardware development.*




## A NEW MODEL FOR DEVELOPING EMBEDDED SYSTEMS

Figure 2 shows an end-to-end development process using modeling and simulation. The system's functional requirements are specified as the inputs and outputs of a model and the restrictions on its implementation. A behavioral model is developed to relate the inputs to the outputs. Simulations of the behavioral model are used to validate that the requirements are satisfied. The model becomes part of the specification, and simulating the model becomes the method for interpreting the specification. Then design of the hardware and software is much less likely to suffer the problems of misinterpretation of an English language description of that behavior. Maintaining the same behavioral description of the system throughout the development process—and filling in the details needed to achieve that behavior at each step in the design, coupled with a co-design and co-simulation environment—allows designers to partition a system into smaller functional elements which, together, retain the overall behavior. Sub-system elements are each described by their own behavior, which is derived from the system-level behavioral model.

This decomposition and stepwise refinement of the behavioral model of the system and sub-systems (and sub-sub-systems) continues until the entire system is composed of functional elements that are small enough that the detailed design of that element is straightforward. Part of this decomposition process also includes partitioning functional elements into hardware and software components which produce the desired behavior of the functional element. This hw/sw partitioning is carried out and determined based on traditional design tradeoffs such as performance, cost, volume, and so on.

The co-simulation environment provides the means to verify the behavior of each functional element composed of hardware and software. The behavior of elements implemented in hardware only is verified by hardware simulation. In all cases, starting from the top level simulation and working down to each functional element of each sub-system, the behavior defined at each sub-system level is used to verify the behavior of the lower level elements making up that sub-system, and so on down to the simulation of the lowest functional elements. Hardware models used in co-simulation, while primarily behavioral models to maintain simulation speed, are developed with the goal of synthesizing that model into silicon. Synthesis of the model (or verification of the model with an equivalent commercial part) and subsequent testing in the system allows the model to be fine tuned, thereby improving the fidelity of the models used to perform simulations.

Finally, by maintaining the behavior that was defined at the highest level of the system down through each subsequent level, the fidelity of the product is ensured. That behavior can be used to establish the tests used for product acceptance.

## HDLS, MODELING, AND SIMULATION

Modern digital design is often performed using hardware description languages such as VHDL (VHSIC hardware description language) and Verilog HDL. Many textbooks about the VHDL and Verilog languages are available.[1,2,3] VHDL and Verilog are similar to programming languages, having their roots in Ada and C, respectively. (Software engineers should take great satisfaction in the observation that hardware design is reduced to programming.) In this section, we will touch on several subjects pertinent to hw/sw co-simulation, including the history of HDLs; how HDL models can describe circuits in different levels of detail to support hierarchical modeling; schematic capture from models; model verification; some different sim-

**LISTING 1**
*The entity, generic, and port statements for an Intel 8051 microcontroller.*

```
entity 8051 is
  generic (tval: 8051delays:=Our_CMOS_delays);
  port(ean, rst, xtal1, xtal2: IN std_logic;
     ale,psen: OUT std_logic;
     p0:INOUT std_logic_vector(7 DOWNTO 0);
     p1: INOUT std_logic_vector(7 DOWNTO 0);
     p2: INOUT std_logic_vector(7 DOWNTO 0);
     p3: INOUT std_logic_vector(7 DOWNTO 0));
end 8051;

architecture RTL_MODEL  of 8051 is
.
  code
.
end RTL_MODEL;
architecture BEHAVIORAL_MODEL  of 8051 is
.
  code
.
end BEHAVIORAL_MODEL;
```

ulation methodologies; and model availability.

VHDL is the older of the two HDLs and was originally developed in the DARPA Very High Speed Integrated Circuit (VHSIC) program as a means of providing precise descriptions of circuitry. As time went on, the HDLs were developed for purposes of modeling, logic design, simulation of models, and synthesis of circuitry from logic designs. HDLs are quite flexible and allow digital components to be represented at various levels of detail. An 8-bit adder, for example, might be described at a behavioral level using an algebraic expression, or it could be described using Boolean expressions. The behavioral description will be quite satisfactory for modeling and simulating many aspects of a system. However, the Boolean description is more readily synthesized into a circuit using an automatic synthesis tool.

Synthesizing circuits from the models used to simulate the design is a very powerful design methodology because of the intimate tie between the model and the final circuit. The model becomes the circuit specification, and simulating the model becomes the method for interpreting the specification. Issues arising from a writer's and a reader's different interpretations of written language are obviated. Top-down design methodologies can begin with high-level behavioral models that are progressively synthesized into register transfer level (RTL) models and, finally, gate-level circuit layouts.

The largest users of VHDL and Verilog are integrated circuit designers. ASICs and gate arrays are commonly designed using these languages, and tools are available for synthesizing models into circuits or for automatically routing connections in the gate arrays. For board-level design, VHDL seems to be the language of choice when an HDL is used. IC designers tend to use a different set of design tools and methodologies than do the IC users.

A high level VHDL description of an Intel 8051 microcontroller is given in Listing 1. The top level of the design is the entity that contains an optional generic statement and a port statement. The entity's name is typically the name of a component. If used, the generic statement contains constants, such as setup times and delay times. The port describes the signals that are the inputs and outputs of the microcontroller. The port statement maps directly to the pins of the device being modeled. In this example, signals are a data type called *standard logic*, which can have nine different values, to account for the electrical properties of real signals. An architecture of the entity is where actual work is done on inputs to produce outputs. We show two different architectures for the 8051. The first, called `RTL_MODEL`, is intended to contain a detailed model suitable for synthesis into an IC. The second, called `BEHAVIORAL_MODEL`, produces the same behavior at the port, with a simpler internal representation. The architectures may be primitives that contain no lower level models themselves, or they may be constructed out of lower level components, which are also entity-architecture pairs.

The structure of the language lends itself well to schematic capture. A symbol for a component can be readily generated from an entity's port statement. The architecture of the entity can be associated with the symbol. Symbols are then linked together into a schematic drawing, compiled, and passed to a simulator. The same circuit can be simulated with different underlying models by changing the architectures used for the symbols.

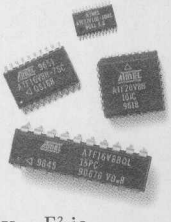
To illustrate the difference in complexity between RTL and behavioral descriptions, consider how registers can be represented and incremented. An RTL description of an 8-bit register will be done using an entity-architecture pair. The register inputs are a data byte signal, clock and clear signals, and an output data byte signal. Their data type will be standard logic. Storing the signals will probably take 18 bytes, one for each bit. In our 8051 model, the architecture is about 15 lines of code with loops, and quite a number of steps need to be performed to move data from the input to the output. To simulate incrementing the RTL register, its contents will be transferred to the accumulator, incremented, and written back to the register, taking hundreds of machine instructions. In contrast, to represent the behavior of a register as seen from outside of the model, it is enough to have a one-byte variable that contains the register contents. Incrementing a register in a behavioral model may take only three instructions: fetch to accumulator, increment, and store to register. This comparison makes several points for hw/sw co-simulation. First, an HDL model may be complex or simple. One needs to ask about the nature of the models being used. Second, the choice of model type can affect the execution time of a simulation by orders of magnitude. Third, one needs to be concerned with the fidelity of the model, such as how accurately the behavior of


**ATMEL OFFERS HUNDREDS OF SPLD TYPES IN 16V8, 20V8 AND 22V10 CONFIGURATIONS IN A BROAD RANGE OF SPEEDS, PACKAGES AND POWERS.** Your job will get easier, and your company will get to market faster and be more profitable, by taking advantage of Atmel's short lead times and low cost. And Atmel's tremendous variety of SPLDs means you can get exactly the right part for the application. For example, choose from packages like our ultra-small and thin TSSOP. Or compare power—you'll find we have the lowest available, as well as speeds that complement your most advanced designs.

| Architecture | Std. Power-Fast | Std. Power | Qtr. Power | Zero Power | Low Voltage | Low Voltage Zero Power |
|---|---|---|---|---|---|---|
| 16V8 | 5,7 | 10,15,25 | 10,15,25 | 12,15 | 10,15 | 15,25 |
| 20V8 | 7 | 10,15,25 | 10,15,25 | | | |
| 22V10 | 5,7 | 10,15,25 | 15,20,25 | 12,15 | 10,15 | 25 |

*Speed in nanoseconds*

We come by all this honestly—$E^2$ is our core technology, we own our own fabs, and we produce some of the smallest dies in the industry. This means we can treat you as well in the future as we do today.



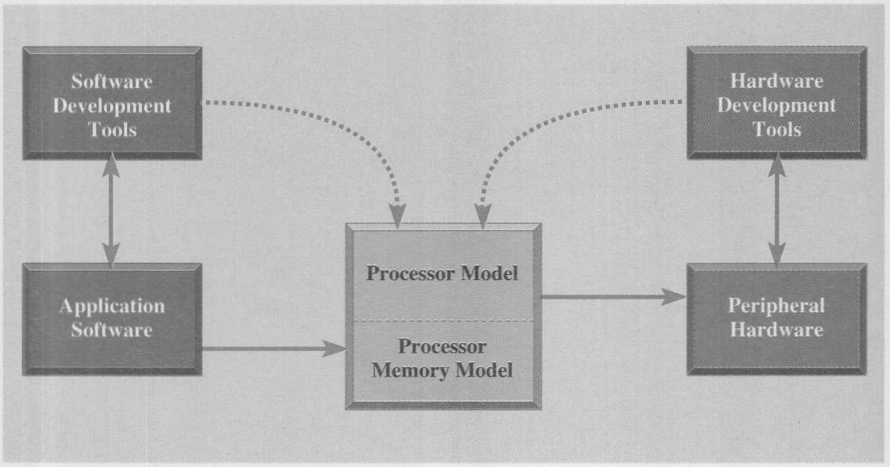
*Get ultra-small and thin TSSOP packages, and extraordinary L and Z performance for power savings and extended battery life.*

Call **1-800-365-3375** to get our new *Configurable Logic Data Book*. And to begin a relationship with a company committed to serving your programmable logic needs for the long haul.

ATMEL

E-MAIL: literature@atmel.com
FAX-ON-DEMAND:
(800) 292-8635 (North America)
(408) 441-0732 (International)
WEB SITE: www.atmel.com
CORPORATE HEADQUARTERS:
2325 Orchard Parkway, San Jose, CA 95131
TEL: (408) 441-0311 FAX: (408) 487-2600

**FIGURE 3**
*Elements of an embedded system in a co-simulation environment.*




a circuit will be modeled. Verifying fidelity of a behavioral model of a register to an RTL model should be trivial, but in more complicated examples, this verification becomes a difficult problem in its own right.

Elementary logic primitives are available with most HDL simulators. Models of common circuits in behavioral and synthesizable forms are available from vendors. Behavioral HDL models of many ICs are commercially available from companies such as Logic Modeling, CAST, and Sand Microelectronics. Models of microprocessors are more difficult to obtain. Synthesizable models used by manufacturers are, of course, valuable intellectual property. Some vendors sell only compiled versions of models. Working with a compiled version of a model limits your ability to see what is going on inside it, a potentially important part of debugging software in a virtual environment. The availability of models may influence how an application is modeled.

HDL simulators are available from several vendors, including Synopsys, Viewlogic, Cadence, and Model Technology, but there are also two major simulation algorithms to choose from. Most HDL simulators on the market today are event-driven simulators. An event-driven simulator keeps

track of events happening at arbitrary times, which is necessary for simulating asynchronous logic. Cycle-based simulators are just being introduced in the market. Cycle-based simulation requires all logic transitions to occur on clock cycle boundaries, such as in fully synchronous logic designs, dramatically reducing the scheduling overhead. Simulation time is reduced by as much as 10 to 50 times for suitable models.

The execution time of simulations is important for hw/sw simulation. Our RTL model of the Intel 8051 executes about 10$\mu$s of simulation time per second of real time using a Viewlogic simulator running on a Sun Ultra 2 (170MHz). In comparison, our behavioral model executes about 1.0ms simulation time per second of real time (and could be five to 10 times faster). Before concentrating on tradeoffs for hw/sw simulation, it is worth briefly considering alternatives to software-based simulation.

**TABLE 1**
*Speed and ease of use of processor models using simulation and emulation.*

| Model Type | Speed (instruc/s) | Control & Visibility | Model Availability | Model Difficulty | Fidelity |
|---|---|---|---|---|---|
| Event Driven RTL | 1-10 | 10 | 3 | 9 | 10 |
| Cycle-Based RTL | ~100 | 10 | 3 | 9 | 9 |
| Event-Driven ISS | $10^4$ | 9 | 6 | 5 | 7 |
| Cycle-Based ISS | $10^5$ | 9 | 2 | 5 | 7 |
| "Processor-less" | $10^3$-$10^5$ | 2 | 6 | 3 | 4 |
| Hardware Modeler | 10-$10^W$ | 3 | 5 | 6 | 6 |
| Gate-Level Emulation | $10^5$ | 5 | 2 | 10 | 9 |

## HARDWARE-ASSISTED MODEL SIMULATION

Simulation time is an issue that is also faced by hardware designers who must verify the correctness of large designs. Hardware designs are verified at the logic-gate level to the greatest extent possible. Verification of large gate-level designs can be addressed by hardware-assisted simulation and by logic emulation.

In hardware-assisted simulation, a high-powered co-processor is used to execute the simulation of the gate-level model. Other parts of the simulation, such as peripheral circuits and test

benches, are left in the software simulator. Zycad is a major accelerator vendor that recently introduced a product called Lightspeed that executes about 4,000 instructions per second for an 80486-class processor, approximately a thousand times improvement over the speed of a software simulation. A Lightspeed accelerator will cost several hundred thousand dollars.

In logic emulation, the entire gate-level design is loaded into a matrix of programmable gate arrays. Quickturn is the largest vendor of this type of product. The major components of a "Quickturn box" are field-programmable gate arrays, the electronics to monitor the arrays, and the software to control the system. The gate arrays can be clocked at 1 to 4MHz, so a model can be executed at speeds that are very fast compared to simulation, 1% to 10% of the full design speed. Logic emulation is considered to be an important part of IC design verification.[4] Maintaining and using an emulation system is also a substantial amount of work, and the price is in the hundreds of thousands of dollars.[5]

*Hardware modeling* is a hybrid technique that runs software on a hardware processor which is interfaced to a software simulator which runs models of the peripheral circuitry. The speed of the entire simulation is controlled by either the software simulation or the communications overhead between the hardware processor and the simulator. Hardware modelers typically execute 10 to 50 instructions per second.[6] However, the microprocessor is much easier to buy than the model the vendor wrote to design it!

So, what is the message for hw/sw co-simulation? If your company's hardware design path uses accelerators or emulators, it will be valuable to debug software on these models, if you can attach a software debugger to them. If not, they are probably inappropriate for hw/sw co-simulation because the accelerators and emulators work with gate-level models that take a lot of work to build and have more detail than necessary for most of the software development.

## PROCESSOR MODELS

Figure 3 shows the major elements of a hw/sw co-simulation environment. The environment is composed of software, a processor model with memory, and peripheral hardware. Note that most processor interactions will be with the program and data memory; at most, a few percent of its activity will be communication with peripherals. The software and hardware development tools allow the developers to view and control the simulation of the software and hardware models.

# Hitachi ICs bring your biggest ideas down to size.

SuperH™
RISC engine



*IC solutions for the hottest applications from 2-way wireless paging to Windows® CE Handheld PCs.*

These days, the biggest ideas in consumer electronics are all about the same size: Handheld. They're now the "Wow" of Wall Street; the "Egad" of editors. And they're the electronics industry offering a hand to millions who have not yet "gone digital." For OEMs of successful Personal Access products, their ever-increasing integration within the size, power and cost constraints of handheld systems is good reason to shake hands with Hitachi.

**How to get bigger, better, smarter, smaller.** Thanks to Hitachi's ability to combine its best-selling line of MPUs, MCUs and advanced memory devices, and deliver these as integrated solutions, we have become the leading IC supplier for handheld systems. In fact, Hitachi's SuperH RISC Engine is the processor of choice for the overwhelming majority of the new Windows CE Handheld PCs.

**Hitachi helps you hit the small time.** To learn how you can get small fast, phone 1-800-446-8341, ext. 800. Or visit our web site at **www.hitachi.com.**

At Hitachi, we understand that the trick is not to think big; the trick is to think big, then to think really, really small!

**HITACHI**

#1 in RISC Shipments

Depending upon the models chosen, the processor and memory may or may not have internal states visible from the outside. The method of modeling the processor and its interaction with memory is one of the most important choices to be made. As one considers the choices, the 20-80 rule (20% of the effort yields 80% of the results) should be kept in mind, and balanced against the possibility that a serious error will be missed. Table 1 shows the execution speed and ease of use for various combinations of emulation methods and processor models and simulation.

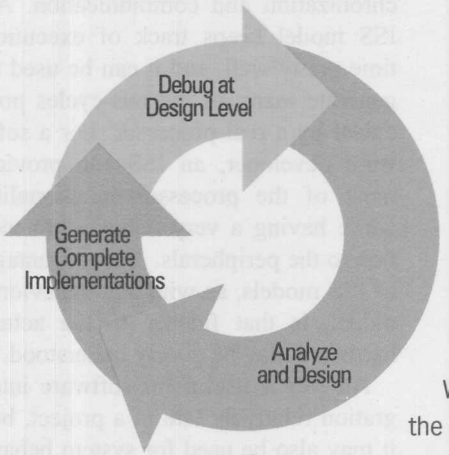
The tradeoffs to be evaluated in this choice are between model and timing accuracy; visibility of the internal state of the processor and peripheral models for debugging purposes; model availability; and simulation speed.[7] How you make tradeoffs depends upon where you are in the design process

**The method of modeling the processor and its interaction with memory is one of the most important choices to be made.**

and the type of design (board vs. ASIC) you are using. Modeling and co-simulation can be used early in a design to evaluate software-hardware tradeoffs that will affect the hardware design—a co-design activity by our definition. For example, early in a design, the required processing power must be established. Later in a design, the processor is fixed, and you may be doing the software-hardware integration in advance of receiving a prototype, what we defined as co-simulation. The objective of the modeling and the types of models that are available for the hardware simulation also depend on whether the design is of a board-level system built with commercial ICs, a board with a commercial processor and custom ASICs for peripherals, or a fully custom ASIC with an embedded processor core. Rework costs become progressively larger with these three options.

At the co-design level, high-level models that run high-level code without showing internal details of proces-

# BE WRONG MORE OFTEN




**Spend more time designing with Rhapsody.**

*Rhapsody*™

**And then, be right when it counts.**

The path to design perfection requires rigorous exploration. And knowing early on what won't work is a sure way to find out what will.

But there's the age-old conflict of embedded systems software design. You have a deadline. Do you take the time to plan, analyze and explore, or succumb to the rush to code?

What if there were a way to "freeze the clock" and take all the time you need to plan and validate your design–actually *see* how it behaves before it's completed? What if you could test every concept? Explore new ideas?

What if there were a way to make a change to your design and *instantly see* it reflected in your implementation? Or make a change to your implementation and *instantly see* it reflected in your design? What if there were a way to do all of this, and get full, optimized and targeted production-quality C++ *automatically?*

The software Holy Grail? Yes. **Introducing Rhapsody, the first UML-based OO software design and implementation tool.** Rhapsody will remarkably change the software development paradigm from time spent coding to time spent designing.

**Save time and money.** Inquire about Rhapsody now. For a special, limited time offer and white paper, "*The Next Generation: Modeling with UML,*" contact us toll free at 1-888-8 ILOGIX (1-888-845-6449) ext. 100, or email info@ilogix.com and get the score on Rhapsody.

REAL-TIME OBJECT-ORIENTED TRAINING SERIES

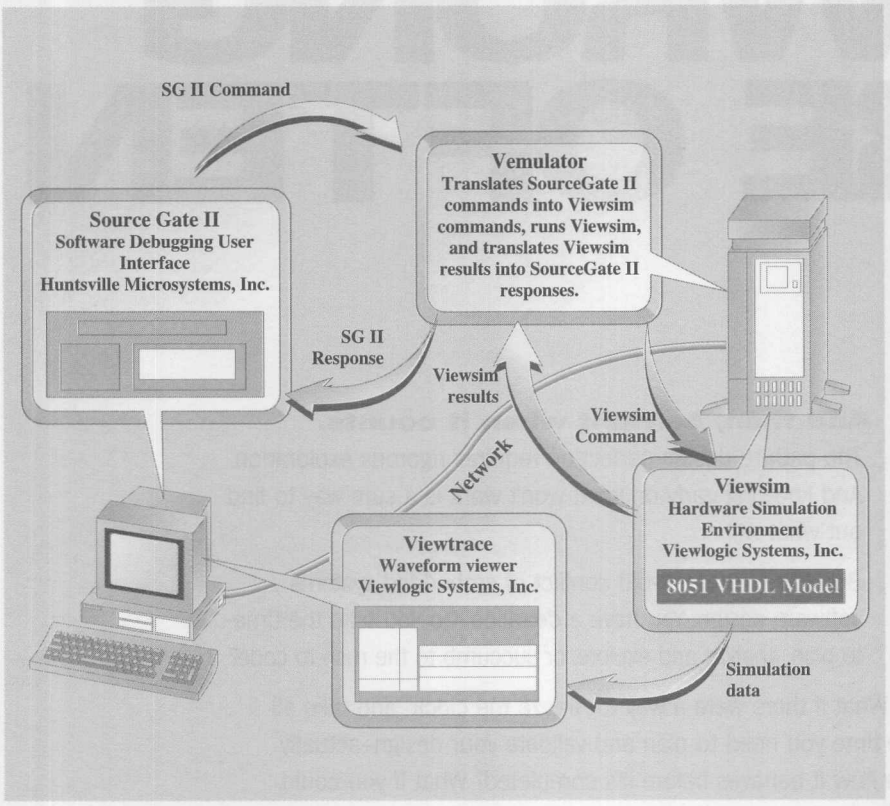
**The only series of its kind: *OO Analysis & Design, and Software Scheduling***

i-Logix

Three Riverside Drive • Andover, MA 01810 • 508-682-2100 • info@ilogix.com • www.ilogix.com

**CIRCLE # 13 ON READER SERVICE CARD**

**FIGURE 4**
*Co-simulation environment constructed from commercial, off-the-shelf tools.*




sor operation may be satisfactory. Some provision needs to be made for the occasional interaction with the peripherals. For board-level design with discrete ICs, you will probably find only behavioral models of the processor and the peripherals. If you use custom ASICs, you may have access to the RTL models used to design them. After receipt of a board-level prototype, you will still have good visibility of the interconnections between the processor and the peripherals, so that conventional methods can still be used for final integration. For an ASIC with an embedded processor, models may be available down to the RTL level for the entire design. In this case in particular, detailed models may be desirable for portions of the hw/sw integration because it will be the only prototype available before a very expensive and time-consuming ASIC turn. Even after the ASIC is fabricated, you will have little visibility into it for debugging; the software and hardware viewers may need to be your in-circuit emulator and logic analyzer. RTL processor models give the best accuracy and internal visibility, but they are difficult to obtain. If the processor and peripherals are asynchronous, event-driven simulation is required to represent their interactions exactly, and only a few instructions per second will be executed. At the cost of only evaluating the simulation at clock-cycle boundaries, a cycle-based simulator increases speed to about 100 instructions per second. The value of using an RTL model is that it will have nearly absolute fidelity with the hardware synthesized from it.

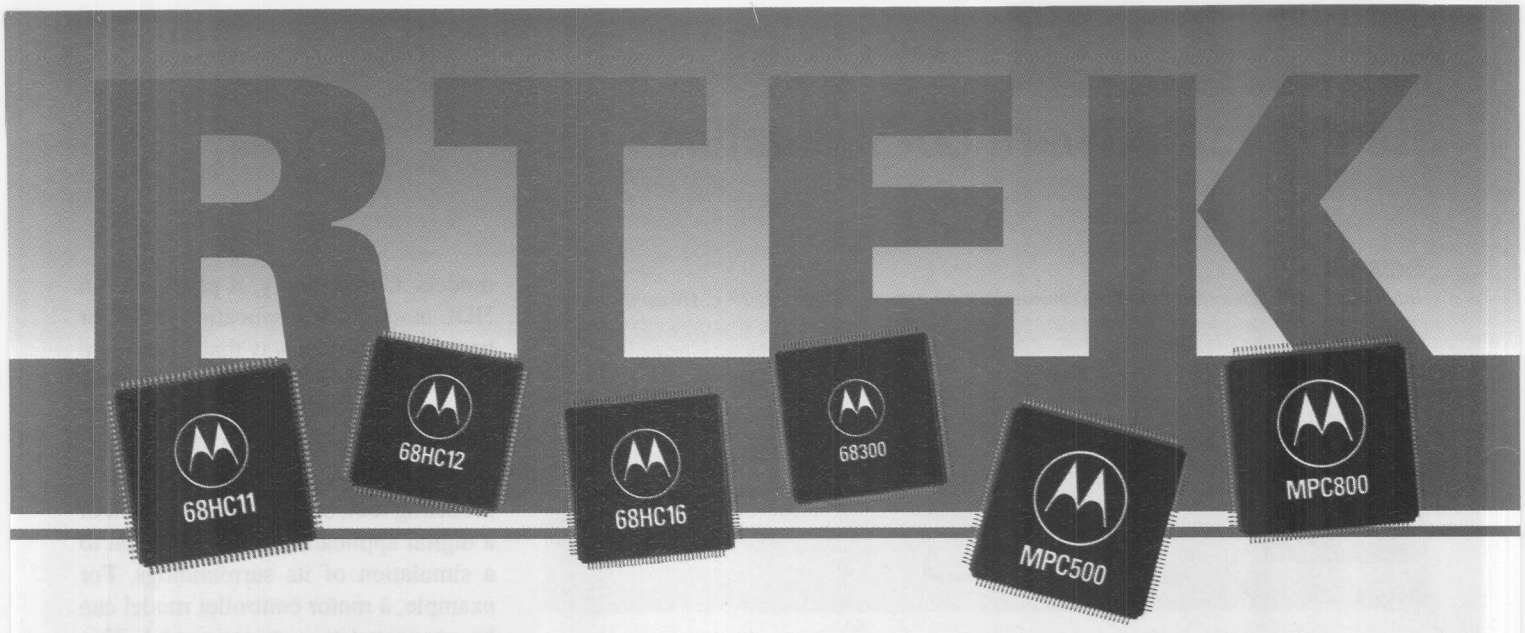
Simplified processor models may be acceptable to the software developer. Instruction set simulator (ISS) models of processors are commonly included in software development environments. An ISS model includes the internal registers and memory, although it probably assumes sequential execution of the instructions, an assumption that may not be satisfied for pipelined processors. An ISS can be written in an HDL and be executed by the HDL simulator with the peripheral models. Execution speeds can be increased to the 2,000 to 20,000 instructions per second range. An ISS can also be written in a conventional programming language and executed in a process separate from the HDL simulator if there are means for synchronization and communication. An ISS model keeps track of execution time pretty well, and it can be used to generate many of the bus-cycles produced by a real processor. For a software developer, an ISS can provide most of the processor functionality while having a very realistic connection to the peripherals. A disadvantage of ISS models, as with any behavioral model, is that fidelity to the actual hardware may be poorly understood.

An ISS is useful for software integration relatively late in a project, but it may also be used for system behavioral modeling earlier in the design. In conjunction with behavioral models of peripherals it can be used to help evaluate processor requirements, hw/sw tradeoffs, and validate system conformance to requirements, before proceeding with detailed design.

Further simplifications are possible as well. The application software could be a program written in a high-level language, executing on a workstation in the workstation's native machine language, communicating with the hardware simulation by special function calls.[8] Synchronizing the software execution with the simulation of the peripheral hardware will be imperfect, but perfect synchronization may not be necessary.

## CO-SIMULATION PRODUCTS

Tools and environments for really doing hw/sw co-simulation began to appear in 1995. An effective environment must tie together tools familiar to both the software and hardware engineers. This environment is difficult to produce because of




# WINNING COMBINATION

## An Unbeatable Team for Your Real-Time Embedded Design.

**It's an unsurpassed combination:** Motorola's broad line of microcontrollers. Plus Motorola's real-time operating system—developed specifically for the 68HC12, 68HC16, 68300, MPC500, and MPC800 Families.

**And now for the 68HC11, too!** Big news. Motorola's real-time operating system can now be used for embedded design on one of the most widely used microcontrollers out there.

**The RTEK™ kernel.** Our optimized kernel can help you use Motorola microcontrollers in winning ways. The RTEK kernel delivers maximum performance with minimum size. It's a field-proven operating system with an easy-to-use C language interface, plus it supports both static and dynamic kernel objects.

**With a new feature: scalability.** The RTEK kernel is fully scalable for most Motorola microcontrollers, such as the 68HC11, 68HC12, 68HC16, and 68300 Families. That can greatly reduce program memory requirements, and decrease your system and component costs.

**Faster time-to-market.** The RTEK kernel is designed to help reduce code development and test time. It features more than 190 kernel services that provide task, memory and interrupt management, event synchronization, data movement, and exclusive accesses. Three separate scheduling methods are supported—pre-emptive, time-sliced, and round robin. The RTEK kernel can be used with confidence because it reflects the same commitment to quality found in Motorola's microcontrollers.

**Call today for a free demo kit.** Find out all that the RTEK kernel can do for you. Just dial **(800) 262-5486 ext. 961** today for more information and to order the free demo copy of the RTEK kernel. Or, visit our web site at **http://www.mcu.motsps.com** to see our full product portfolio.

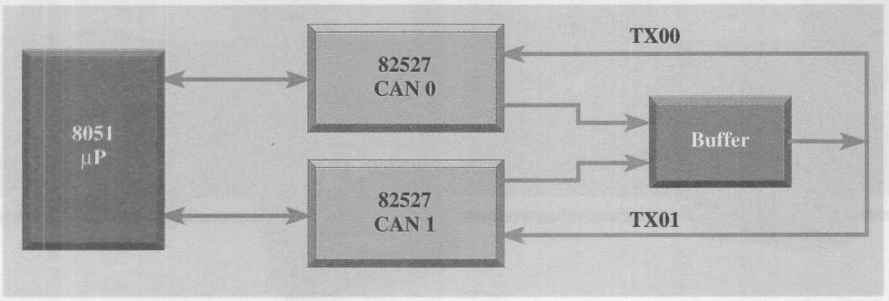
**MOTOROLA**

*Microcontroller Technologies Group*



*What you never thought possible.™*

**FIGURE 5**
*Intel 8051 initializes and handles communication between Intel 82527 CAN controllers.*




the many tradeoffs to be made, and no single right way accomplishes the task. We are aware of four companies that provide capabilities for hw/sw co-simulation. In this section, we give brief descriptions of their products, as we understand them.

## HIGH-LEVEL MODELING: EAGLE DESIGN AUTOMATION

Eagle Design Automation offers two products that address sw-hw co-simulation: EagleI and EagleV.[9,10,11] Both are relatively high-level modeling tools, with two major components. The first is the Virtual Software Processor (VSP), and the second is the Virtual Product Console (VPC).

The Virtual Software Processor executes high-level language code compiled into the native machine language of the workstation. The only internal behavior of the target processor and its memory that is modeled is the part that handles interactions with external devices, such as I/O ports. High-level language function calls communicate through these ports with the peripheral device models that are executed by an HDL simulator. The strengths of this method are the simplicity of the model and rapid execution speed (it is capable of running at 5,000 instructions per second). The method has some weaknesses: model verification is difficult because the models are far-removed from the detailed behavior of the target processor and execution of the software and the hardware simulation are difficult to synchronize. Three methods are used for synchronizing the VSP with the HDL simulator.[12] The first is Polled Status synchronization in which a hardware action is initiated by the software, which then polls a status register until a flag is set signifying that the hardware action is complete. The VSP can also respond to "hardware" driven interrupts much as a hardware processor would respond. The third way is for the software to estimate how long a simulation of a hardware action will take.

The Virtual Product Console links together the software development environment, the VSP, the hardware simulator, and its development environment. Commercial software development tools are supported by the VPC. The VPC connects the VSP to VHDL and Verilog simulators from most of the major CAE vendors. One of the virtues of EagleI is that it should fit into many existing design environments. Another interesting feature is that the VPC can link multiple processor models to the HDL simulator.

In their Eagle Design Application Note, "An Evolution in System Design," G. Zach and J. Wilson describe use of an EagleI prototype in the design of a complex ASIC.[12] Three serious errors were caught in simulation which would have resulted in ASIC redesign and re-spin.

## VIRTUAL ICE: YOKOGAWA ELECTRIC CORP.

Yokogawa offers products called Virtual ICE and VMlink.[13] Virtual ICE provides an in-circuit emulator environment for code development on HDL models. Conceptually, it plugs into an HDL model of an application (ASIC or board) in the same way that a hardware ICE plugs into a processor socket in a circuit board. VMlink is an interface for connecting to the MATLAB simulator. MATLAB is a more general modeling tool, so that the simulation of a digital application can be coupled to a simulation of its surroundings. For example, a motor controller model can be connected to a motor model. This tool looks very useful if it will fit into your design environment.[14]

Virtual ICE consists of a processor model and an ICE interface to the processor. The processor model is a bus-cycle-accurate ISS written in Verilog. Yokogawa's method for model verification is not disclosed. The present version of Virtual ICE can be run using the Verilog-XL and VCS Verilog simulators. Future versions of Virtual ICE will run under "bilingual" simulators that will accept designs with a mix of VHDL and Verilog components. Off-the-shelf processor model offerings are somewhat limited, but custom models are available. The Verilog language includes a programming language interface, which is used to connect to ICE software. The ICE software provides windows for source code, tracking assembly language, internal registers, memory, and breakpoints. Execution of the model is controlled using the ICE interface. We have seen little emphasis on visibility into the hardware side of the design, but presumably there is some access through the Verilog simulator.

## SYSTEM LAB AND BEHAVIORAL VERIFICATION TECHNOLOGY: CPU TECHNOLOGY

CPU Technology's main business is design of custom microprocessors, for which they have developed their own design methodology and proprietary tools. The design of MIPS quad-processor systems with this approach appears to be one of the most advanced top-down design methodologies in use today.[15]

They offer two products, SystemLab and Behavioral Verification Technology (BVT), but CPU Technology does not make their language or simulators available for sale. What they do offer is a modeling service that provides their customer a model and the environment for working with it. They work with customers to take models to silicon.

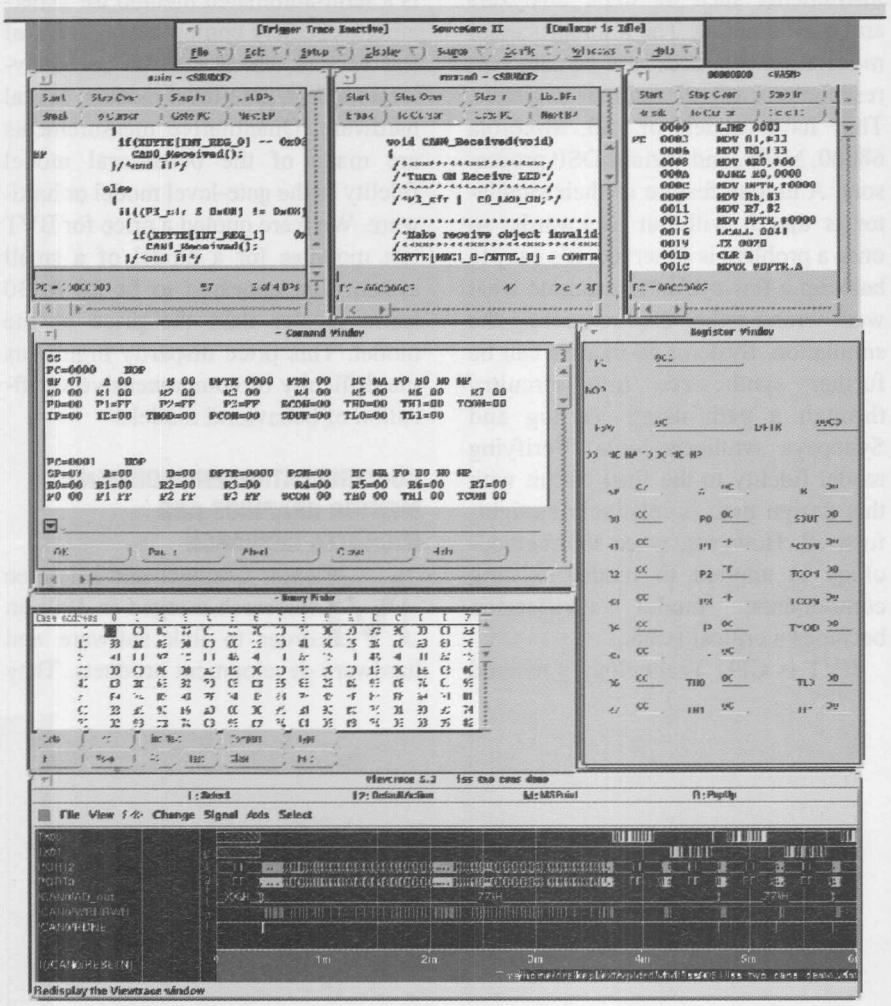
SystemLab is an environment for developing behavioral models of systems and their components. Models of processors and peripherals are constructed using CPU Technology Design Language, a proprietary HDL. The models are executed using a proprietary cycle-based simulator. Their design approach is to design the functionality of all system components at this behavioral level before going to detail design of the components themselves. Very high visibility into designs is available using virtual instruments such as logic analyzers and oscilloscopes. The microprocessor models run real code and contain representations of all internal registers. They have models of x86, Motorola 68000, MIPS, and various DSP processors. A unique feature of their simulator is that it will run backwards, so once a problem is observed, it's easy to back up a few cycles to examine what went wrong rather than re-starting the simulation. SystemLab models can be further synthesized into circuitry through a path using Verilog and Synopsys synthesis tools. Verifying model fidelity to the final circuit with this design path is relatively straightforward. However, when the methodology is applied to model existing components, model verification becomes a critical issue.

BVT is CPU Technology's method for verifying the accuracy of models. It is a semi-automated method for generating extremely comprehensive sets of test vectors which can run on behavioral models, gate-level models, or real hardware. Quantitative measurements are made of the behavioral model fidelity to the gate-level model or hardware. We were quoted a price for BVT test modules for a model of a small system that appeared to be 20 to 30 times greater than the price of the model. This price disparity highlights the difficulty of comprehensive verification of behavioral models.

### CO-VERIFICATION ENVIRONMENT: MENTOR GRAPHICS AND MICROTEC RESEARCH

Mentor Graphics and Microtec Research merged in 1995 in order to link software and hardware development products. They

**FIGURE 6**
*EMME's view of the VHDL model's operation with SGII and Viewsim.*



have recently announced a product called Seamless Co-Verification Environment (Seamless CVE).[16,17]

Seamless CVE combines the Microtec X-Ray software development environment with Mentor Graphics' HDL simulation. Microtec has provided instruction set simulators for many processors for some time. Standalone execution speeds of up to 100,000 assembly instructions per second are quoted for the ISS models. Seamless CVE is an interface between the ISS model and the HDL simulator that executes models of the peripheral devices. The interface functions include synchronization of the ISS with the HDL simulator, memory management functions, and transferring information between the ISS and HDL processes. Seamless CVE allows its user to take advantage of the observations that most CPU activity is with memory, and that less than 1% of the instructions communicate with peripherals. The architecture of CVE allows the user to run memory models in the HDL simulation, or in the CVE. They quote execution of 450 instructions per second with memory fetches from the HDL simulation. CVE allows the user to choose (dynamically during execution) various levels of filtering of the transactions between the ISS and the HDL simulation. The first filter is instruction fetch masking. Additional clock signals can be suppressed by the user when he knows that there is no processor-peripheral interaction taking place. With full filtering, execution may speed up to as much as 15,000 instructions per second. This methodology should give good timing fidelity. The fidelity of the ISS to a real processor should still be verified.

## EMBEDDED MICROPROCESSOR MODELING ENVIRONMENT (EMME)

We have been experimenting with ways to provide a co-simulation environment using commercial off-the-shelf tools for over a year. We thought this was an important capability that would shorten our development cycle; improve the integrity, reliability, safety and security of our systems; and provide an excellent spring board to the next major hurdle, a co-design environment. Although when we began, no tools were available, several have become available since then. Coupling commercially available tools allowed us to take advantage of each tool's strengths, providing a kind of synergism not typically found in a single vendor solution to a multi-discipline problem. This gave us the greatest flexibility to configure the co-simulation environment, allowing us to decide what level of detail is needed based on which stage of the development cycle we are in.

Figure 4 shows the Embedded Microprocessor Modeling Environment (EMME) we developed at Sandia. There are four software components running on two workstations on a network. The computer in the lower left of the figure displays the GUIs for the hardware and software and the computer in the upper right is a fast, high powered workstation running the VHDL simulator. All of the elements will run on a single workstation without modifications, or, as shown, coupled across a network, allowing additional computing power to be utilized. The first component is a software debugging tool called SourceGate II (SG2) from Huntsville Microsystems, Inc., which is the GUI

# For a limited time,*

# you can get up to 65% off the purchase

# of Texas Instruments DSP Development Tools.

| | C2x | C2xx | C3x | C4x | C5x | C54x | C8x | Host |
|---|---|---|---|---|---|---|---|---|
| Assembler/Linker | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | PC |
| C Compiler/Asm/Linker | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | PC/SPARC |
| Debuggers | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | PC/SPARC |
| Evaluation Module | ✓ | | ✓ | | ✓ | ✓ | | PC |
| Simulators | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | PC/SPARC |
| Emulation Hardware | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | PC/SPARC |
| Digital Filter Design Package | ✓ | | ✓ | ✓ | ✓ | | | PC |
| JTAG Emulation Cable | | ✓ | | ✓ | ✓ | ✓ | ✓ | PC/SPARC |
| MPS D Emulation Cable | | | ✓ | | | | | PC/SPARC |
| Code Generation Toolset | | | | | | | ✓ | PC |
| SDB | | | | | | | ✓ | PC |
| S/W Toolkit | | | | | | | ✓ | SPARC |
| PPDS | | | | ✓ | | | | PC |

Marshall

It's about time

TEXAS INSTRUMENTS
Authorized Distributor

To order: call 1-800-877-9839 ext 1565

http://www.marshall.com/dsptools.htm

*Limited time offer good through June 30, 1997 for products listed.

CIRCLE # 15 ON READER SERVICE CARD

for their in-circuit emulators (ICE). The second component is the Powerview framework from ViewLogic Systems, Inc. We use the Viewsim VHDL simulator and the Viewtrace waveform viewer for real-time display of hardware signals, and we use other tools for schematic capture and model compilation. The third component is our command translator, Vemulator. The fourth software component is a VHDL model of an Intel 8051, for which we used two different models. We used both a synthesizable RTL model from Sandia's digital ASICs department and an instruction set simulator written in VHDL by Gary McGovney.

Our work centered on Vemulator, which executes on the same workstation as Viewsim. Vemulator starts and initializes Viewsim, Viewtrace, and the VHDL models. Vemulator is a server for SG II, accepting ICE commands, translating them into Viewsim commands, gathering the results of the simulation from Viewsim, and returning a response to SG II.

We applied EMME to a simple circuit shown in Figure 5. The 8051 runs a program which initializes the two Intel 82527 CAN controllers, and then controls the exchange of messages between them. Figure 6 shows EMME's view of the VHDL model's operation with SG II and Viewsim. The upper windows are SG II windows showing the C main program, a C subroutine, assembly code, a command window that was used to single step assembly code, memory, and register maps. A Viewsim window at the bottom shows 6ms of simulation time during which the CANs were initialized and exchanged two message packets. There are four signals between the circuit components, as well as three signals internal to the CAN controllers. It is more than a conventional software simulation view because the user can observe any hardware signal through the Viewsim window, even internal signals which would not even be visible in actual hardware. This enormous increase in signal visibility, before building a prototype, is where co-simulation offers an advance over older design methods.

This simulation took about 10 minutes of actual time to execute. Of this, about six seconds were used to execute the ISS processor model. Adding the two CAN controller models to the circuit slowed execution by almost 30 times. Displaying 33 signals (only seven are shown) in Viewtrace as they were generated slowed the simulation by another four times. Using the RTL 8051 model only slowed the simulation another 30%, although it is 100 times slower than the ISS model. An important point to remember when evaluating products on the market is that even with a fast, simple processor model, executing peripheral logic models (your design that you really care about) and display software may very well be your real source of performance limits.

We think this approach is valuable because, for the price of writing the translator program (less than 10,000 lines of code), we retained the development environment already familiar to software and hardware engineers and added powerful new capabilities while maintaining flexibility. Developing Vemulator allowed us to couple a tool used by software developers for hw/sw integration to the hardware simulation environment used by our hardware designers. This allows the two sides to work concurrently while keeping their accustomed development environments. Existing tool knowledge and experience is maintained throughout the development process and training for similar tools used at different stages is kept to a minimum. An even more important advantage to us is the ability to change the level of detail contained in the hardware models at various stages in the development process. At the front end of the development effort, fairly simple behavioral models and an ISS (written in VHDL) for the processor can be used to define the system level behaviors. Later, when more detailed design information is available, simpler models can be replaced by the more detailed models, allowing the checkout of critical interfaces and high consequence elements of the system like security critical functions. The design team, rather than the tool, is allowed to make the decision as to the amount of detail necessary.

## INCREASING CONCURRENCY

The key to reducing the time to market for embedded processor systems is to adopt development methodologies that greatly increase the degree of concurrency of software and hardware development. We see hw/sw co-design and co-simulation as two elements of the new paradigms. The technology of hardware modeling and simulation has advanced to a stage at which software can be sensibly developed on models of hardware, the co-simulation part of a new paradigm. A key to successful modeling and simulation is to make models with no more detail than necessary and to choose an appropriate simulation method. Many ways exist to construct sw-hw simulation environments. The trick will be to identify methods that will improve productivity at your company. ESP

*We wish to thank Huntsville Microsystems for their assistance with interpreting SourceGate II commands, Viewlogic Inc. for help with coupling to the Powerview environment, Sandia's Digital ASICs Department for providing the RTL model of the 8051, Gary McGovney for the ISS model of the 8051, and Dale Brandt for the code in the CAN controller demonstration. We also had valuable demonstrations and conversations with representatives of Eagle Design, Yokogawa Electric, Mentor Graphics and CPU Technology. This work was supported by the U.S. Department of Energy under contract DE-AC04-94AL85000.*

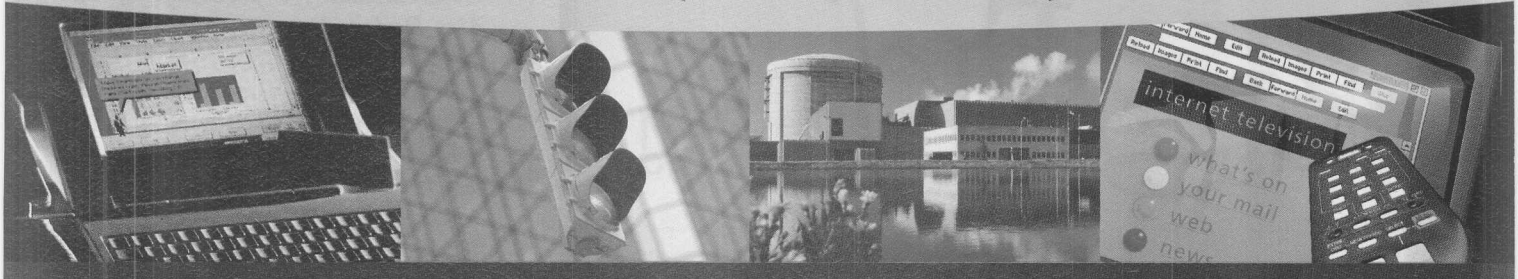
From jet planes to Chunnel trains, QNX goes the distance: testing turbines, loading cargo, even controlling air traffic.

QNX helps healthcare applications run faster, and cost less, thanks to its executive-class speed and full x86 support.

Using a QNX-based vision system, shuttle astronauts stay focused on important things. Like launching satellites.

QNX serves up reliable service at the point of sale, whether it's handling fast-food orders or checking credit cards.

# The QNX realtime operating system is found in these worldwide locations




With its tiny, full-featured Internet suite, QNX helps web-transaction appliances do big business.

From traffic control to process control, QNX drives thousands of mission-critical applications nonstop, 24 hours a day.

When safety is measured in microseconds, nuclear power stations count on QNX: it's a *real* realtime OS.

Thanks to QNX, the web is coming to your living room faster than you can say "URL."

## So many applications. So many demands. How does QNX do it?

Start with rock-solid OS technology field-tested for over 15 years. Add in innovative products like the award-winning Photon microGUI®, QNX's embeddable windowing system. Provide a rich, robust toolset so developers hit the ground running. And keep the memory footprint exceptionally small to ensure runtime costs stay exceptionally low.

Most important, make it all fully scalable. That way, developers can deliver everything from web phones to factory-wide control systems—using a single OS

QNX®

*The Leading Realtime OS for PCs*

**Available Now:**
- internet toolkit
- embeddable GUI & browser
- POSIX & Win32 APIs
- embedded filesystems
- memory protection
- fault-tolerant networking
- distributed processing
- multilingual support
- unrivalled x86 support
- embedded OEM pricing

**www.qnx.com**

**call 800 676-0566 ext. 2344**

**CIRCLE # 16 ON READER SERVICE CARD**

QNX Software Systems Ltd., Voice: 613 591-0931 Fax: 613 591-3579 Email: info@qnx.com
**Europe:** Voice: (44)(0)1923 284800 or 613 591-0931 Fax: (44)(0)1923 285868 Email: QNXeurope@qnx.com

*Philip Dreike and James McCoy are Senior Members of Technical Staff at Sandia National Laboratories in Albuquerque, NM, where they have worked for more than 10 years. Phil and Jim have collaborated for the past two years on this hardware/software co-simulation project. Phil can be reached at dreikepl@sandia.gov and James can be reached at jamccoy@sandia.gov.*

**REFERENCES**

1. Armstrong, J.R. and F.G.Gray. *Structured Logic Design with VHDL.* Englewood Cliffs, NJ: Prentice Hall, 1993.

2. Ashenden, P.J. *The Designer's Guide to VHDL.* San Francisco, CA: Morgan Kaufmann, 1996.

3. Thomas, D.E. and P.R Moorby. *The Verilog Hardware Description Language, Third Edition.* Boston, MA: Kluwer Academic Publishers, 1996.

4. Vivolo, L. A., "Quickturn Design Systems, Inc.," *EE Times*, Jan. 15, 1996.

5. Dickerson, W., "IC Emulation Technology a Mixed Bag", *EE Times*, May 13, 1996.

6, 7. Rowson, "Hardware/Software Co-Simulation", *Proceedings of the 31st Design Automation Conference*, 1994.

8. Wilson, J., "Hardware/Software Selected Cycle Solution," *Third International Workshop on Hardware/Software Codesign*, sponsored by IEEE, Grenoble, France, 1994.

9. Bunza, "Hardware/Software Co-Design, Integration and Verification: Experiences on the Bleeding Edge of Technology," *Proceedings of Embedded Systems Conference East 1996.*

10. www.eagledes.com: Eagle Design Web site.

11. Williams, B., "Software Attacks Co-Simulation Problems," *EE Times*, May 13, 1996.

12. Zach, G. and J. Wilson, "An Evolution in System Design," *Eagle Design Application Note.*

13. www.yokogawa.co.jp/Eda/eda_ice: Yokogawa Electric

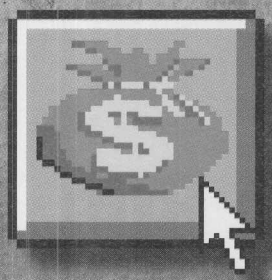
14. Satoru, N., S. Katsuyuki, S. Naoki, and K. Norio, "Fieldbus Communication Controller Chips Developed Using Our 'CEEDS-ASIC' System," *Yokogawa Technical Report English Version*, No. 19, 1994.

15. E. King "The Virtual Lab for Complex Embedded Systems Development," *Proceedings of Embedded Systems Conference East 1996.*

16. Leef, S. and R. Klieg, "Hardware, Software Join in Simulation," *EE Times*, June 3, 1996.

17. www.mentorg.com/products/seamless/: Mentor Graphics Web site

# Click, Click, Click, Done.

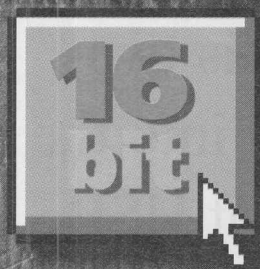


**MQX Royalty free kernel**

- Significant cost savings
- Single and distributed processor operation
- Scalable microkernel architecture
- Extensive embedded I/O components


**32 bit tool support**

- 68xxx/683xx family
- R3000/4000
- ColdFire® and Power PC® coming soon




**16/24 bit tool support**

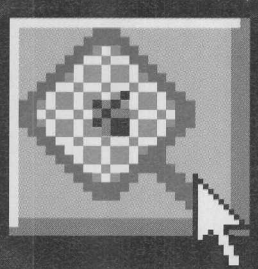
- DSP56xxx
- C166/ST10
- 196/296
- XA
- TLCS-900

You are under constant pressure to get the job done. From start to finish, TASKING tools help you get your embedded application written and tested faster.

Our EDE development environment provides a rapid edit-compile-debug process. Our scalable Precise/MQX® kernel is royalty free. Our compilers produce outstanding code efficiency and our CrossView Pro debugger is a snap with its easy to use interface, code coverage and profiling.

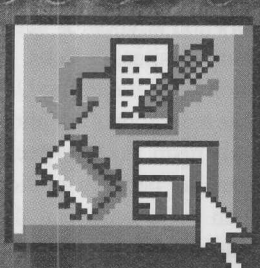
No wonder TASKING is the world's leading choice for software development tools across industry standard computing platforms: Windows®95, Windows®NT, Unix.

To find out how to get your embedded application DONE faster call us for your free demo 1-800-458-8276, or visit us at http://www.tasking.com



**CrossView Pro debugger**

- Full trace including C, assembly and stack
- Kernel aware debugging
- I/O simulation
- Code coverage and profiling
- Multiple execution environments



**Compiler environment**

- Point and click interface
- Totally integrated compiler, assembler and linker/locator
- Automated build process
- Target specific extensions
- Outstanding code quality


**8 bit tool support**

- 251/8x930 USB
- 8051
- SMC88
- HC08

TASKING

Quality Development Tools Worldwide

*BSO/TASKING and Intermetrics Microsystems have joined to become TASKING*

Asia +81 3 3457 6831

U.S. 617-320-9400

Europe +31 33 455 8584

**CIRCLE # 18 ON READER SERVICE CARD**

by JOHN CANOSA

Nance Paternoster

# Understanding Universal Serial Bus Part 1: USB Basics

## Connectivity is a big issue for embedded systems today, and the universal serial bus promises to simplify the process. This two-part series will take you into the underlying layers of what USB technology is all about.

Today's embedded systems designers are finding that more and more of the products they are designing are not isolated standalone devices. We are in the information age, but information is only valuable when it can be accessed and utilized. Connectivity is a key issue for embedded systems today. Being able to connect and share information with a PC-oriented world is of paramount importance. One of the key mechanisms for connecting to PCs and, most likely, workstations and Macintoshes, will be the universal serial bus (USB).

USB is a peripheral bus standard developed by PC and telecom industry leaders—Compaq, DEC, IBM, Intel, Microsoft, NEC, and Northern Telecom—that will bring the plug-and-play of computer peripherals outside the box, eliminating the need to install cards into dedicated computer slots and reconfigure the system. Personal computers equipped with USB will allow computer peripherals to be automatically configured as soon as they are physically attached—without the need to reboot or run setup routines. USB will also allow multiple devices (up to 127) to run simultaneously on a computer, with peripherals such as monitors and keyboards acting as additional plug-in sites, or hubs.

The motivation for developing USB was originally as a computer telephony connection mechanism. As the specification developed, it became obvious that it was much more universal than for just one application. USB was designed for ease of use, incorporating both hot plugging and software plug-and-play mechanisms. It also was designed to allow for easy expansion. USB is a low- to mid-speed serial communications bus that operates at 12Mbps, with a 1.5Mbps sub-channel for cost sensitive applications. Many people see USB as a competitor to

**We are in the information age, but information is only valuable when it can be accessed and utilized.**

another serial bus, IEEE 1394 (Firewire). This really is not the case: the two buses are complementary, with USB occupying the low- to mid-speed range, while 1394 is targeted toward high-speed applications. Figure 1 illustrates the complementary nature of the two buses.

This two-part article investigates USB from the perspective of an embedded systems developer designing a device that acts like a USB slave. This part contains an introduction to USB, concentrating on the peripheral side of the specification. The second part will contain an overview of some of the USB controllers that are available, as well as discuss some implementation issues regarding bandwidth and throughput.

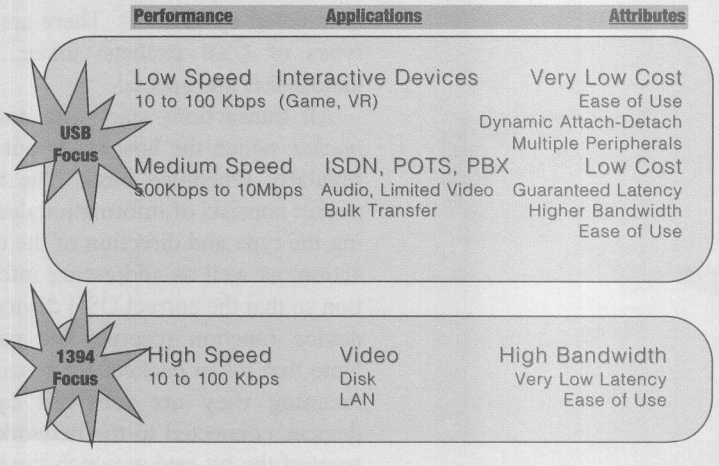
The USB specification describes the mechanical and electrical attributes of the bus, the protocol, the different types of bus transactions, bus management issues, and the upper level programming interface. It is defined to be truly an open standard, so that devices from different vendors will interoperate without any help from the end user.

The following sections of this article will examine the protocol, the four types of bus transactions, and the programming interface, as well as a brief introduction to the physical layer.

## USB ARCHITECTURE

USB is a master/slave type bus. In USB parlance, a host is responsible for performing the scheduling of the bus, and various devices, which can be either hubs or functions that share the serial data stream. The actual physical topology

**FIGURE 1**
*USB and IEEE 1394 focus areas.*



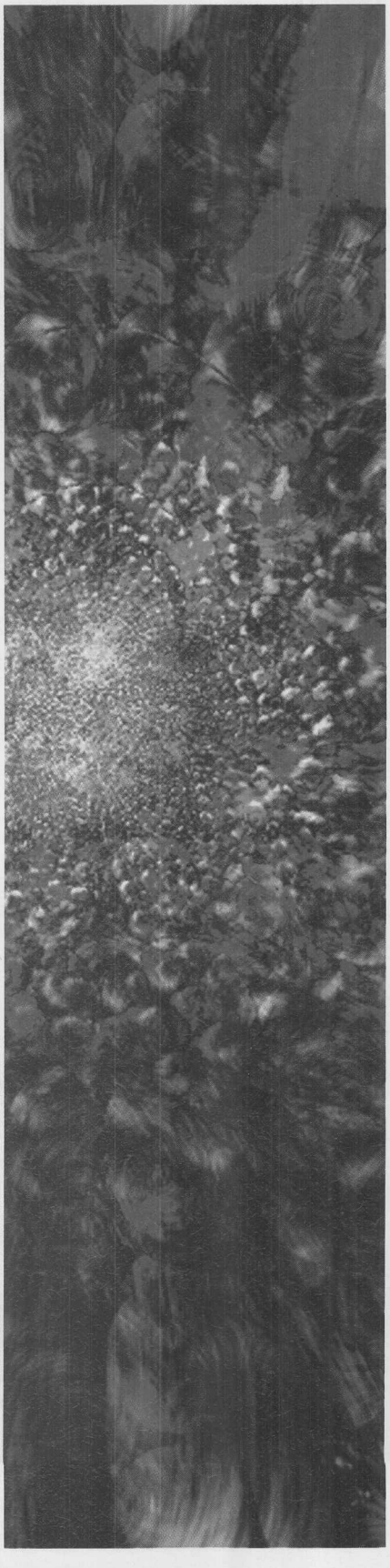



**FIGURE 2**
*Physical hardware view.*

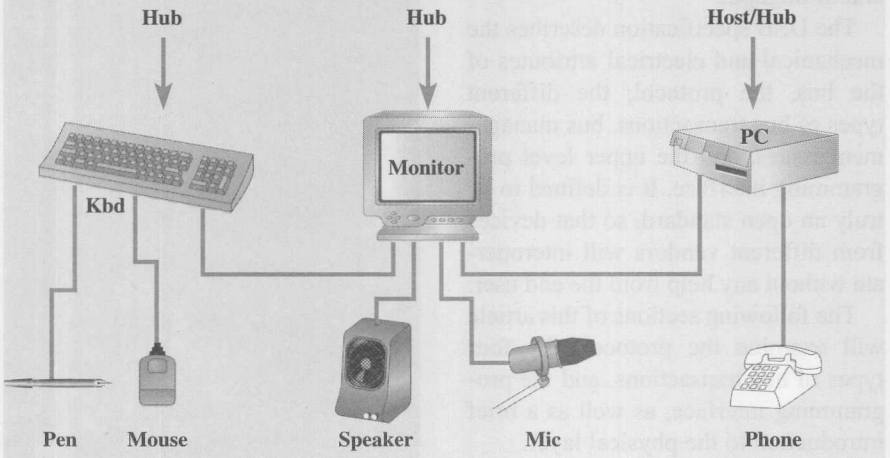
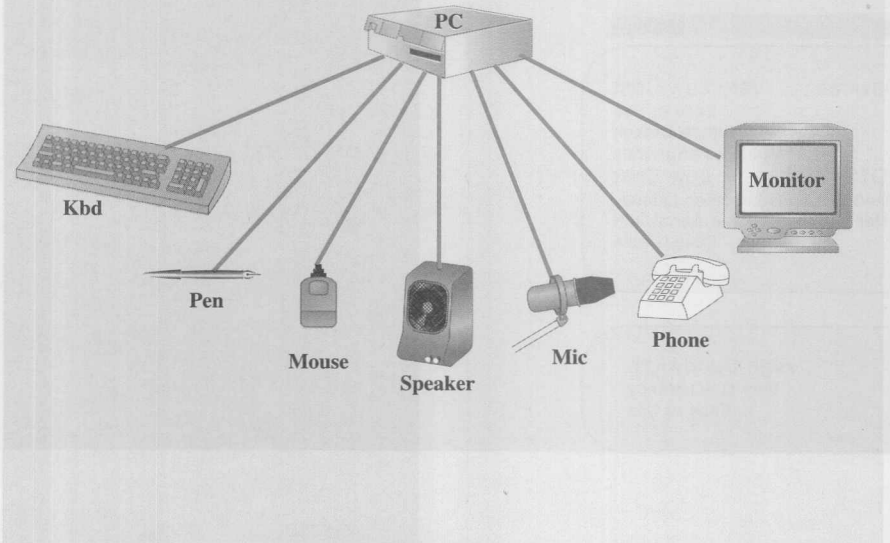



**FIGURE 3**
*Logical hardware connection.*




of the bus is best described as a tiered star network as shown in Figure 2.

All devices are connected over a 4-wire cable, no longer than five meters. Two of the wires are twisted pair carrying differential data, while the other pair of wires provides a limited amount of power so that devices may be powered off the bus itself. Data is transferred using non-return to zero inverted (NRZI) encoding at 12Mbps. There is also a 1.5Mbps sub-channel that can be simultaneously supported on the same wire as the 12Mbps channel. Each device must plug into a special type of USB device called a hub.

Hubs are intelligent devices that not only act as signal repeaters and routers, but also provide some basic power management. Hubs are essential to the plug-and-play implementation. Each hub has the ability to control the power applied to devices connected downstream from it, as well as notifying the host when a device has been connected or disconnected from the network. Once a device is connected to the network and completes the enumeration process (plug-and-play discovery), the hubs act as repeaters and become transparent from a data flow perspective. This is shown in Figure 3.

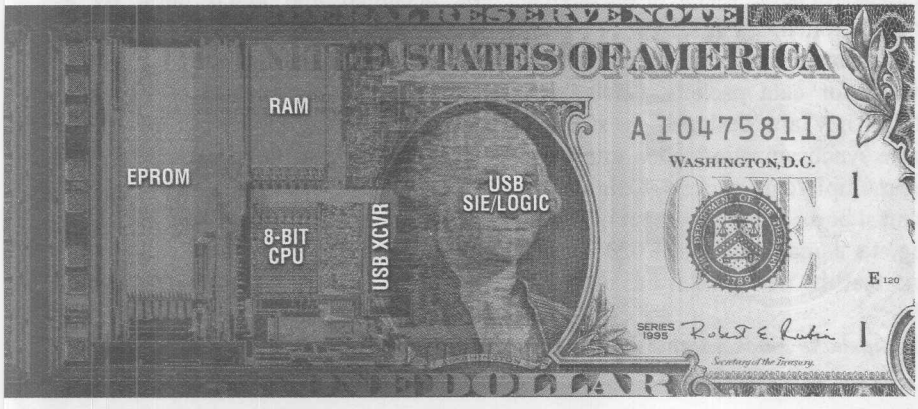
## USB PROTOCOL: PACKET TYPES

All USB bus transactions involve transmissions of one to three packets. Each transaction is initiated by the host. There are four types of USB packets: token, data, handshake, and special.

All transactions start with a *token packet*, which the host sends out on a regularly scheduled basis. The token packet consists of information describing the type and direction of the transaction, as well as addressing information so that the correct USB device and device function receives the packet. Note that token packets are broadcast, meaning they are received by all devices connected to the network that support the bit rate at which the token packet is transmitted.

As shown in Figure 4, each token

# MAKING USB UNIVERSAL: THE FIRST USB µC UNDER $1.00




### 8-bit USB Microcontrollers for Under $1

Cypress's new family of 8-bit microcontrollers offers the lowest-cost solutions for USB peripherals. These highly integrated USB microcontrollers are built on the industry's smallest RISC core that translates into high volume production with pricing under $1.00.

### Full-Feature USB Microcontrollers

Cypress's USB microcontrollers are the first to offer EPROM programmability for fast, easy code customization to speed time-to-market. The USB microcontrollers integrate data RAM, program EPROM, a USB Serial Interface Engine (SIE), and a transceiver for reduced cost and parts count while clock doubler and Instant-On Now (ION™) features ensure low EMI and 70% lower power consumption.

### Optimized for a Full Spectrum of USB Applications

Cypress's USB microcontrollers are ideal for all USB peripheral applications, such as mice, joysticks, gamepads, and keyboards. No matter what USB peripheral you're designing, these low-cost microcontrollers make it easy to implement.

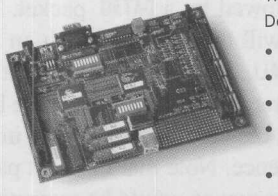
### USB Microcontroller Family

| DEVICE | MAX. RAM | MAX. EPROM | NUMBER OF I/Os | PIN/PACKAGE |
|---|---|---|---|---|
| CY7C630xx | 128 Bytes | 4 KB | 12 | 20-pin DIP<br>20-pin SOIC |
| CY7C631xx | 128 Bytes | 4 KB | 16 | 24-pin SOIC |
| CY7C632xx | 128 Bytes | 4 KB | 10 | 18-pin DIP |
| CY7C634xx | 256 Bytes | 8 KB | 31 | 40-pin DIP<br>48-pin SSOP |
| CY7C635xx | 256 Bytes | 8 KB | 39 | 48-pin SSOP |



### Low-Cost Development System for Fast Time-to-Market

Cypress now offers the CY3650 USB Developer's Kit for $495 (a $1500 value). The kit includes a full-speed hardware emulator to help you develop firmware and system drivers for your USB application.



now $495!

The easy-to-use ~~$1500~~ CY3650 USB Developer's Kit includes:

- In-Circuit Emulator board
- Assembler
- Debugger/Monitor
- Software for mouse and joystick applications
- USB code library

### Start Your USB Design Today

Call now to get the USB Literature Kit, including a copy of the USB specification, a Cypress Data Book CD-ROM, and an order form for the USB Developer's Kit.

**(800) 858-1810**
**FAX: 1-408-943-6848**
**Ask for Kit #T031**

**http://www.cypress.com/cypress/whathot/hot_top.htm**

CYPRESS

**FIGURE 4**
*Token packet fields.*

| | 8 bits | 7 bits | 4 bits | 5 bits |
|---|---|---|---|---|
| Token | PID | ADDR | ENDP | CRC5 |

**FIGURE 5**
*Data packet fields.*

| | 8 bits | 0-1023 bytes | 16 bits |
|---|---|---|---|
| Data | PID | DATA | CRC16 |

packet consists of an 8-bit packet identifier (PID) field that currently have any one of the following:

- IN: signifying a read from a device
- OUT: signifying a write to a device
- SOF: indicating the start of a USB frame
- SETUP: indicating that a command/control data packet will be forthcoming and cannot be ignored

The address field follows the PID. It is a 7-bit field, which effectively limits a USB network to 127 attached devices. After the address field comes the endpoint field. Endpoints are sub-addresses within a particular function of a device and are the actual data source and sink within that function. Endpoints will be discussed in more detail later. Note that in an SOF token packet, the 11 address and endpoint bits are replaced with an 11-bit frame number field. Finally, all token packets end with a 5-bit CRC.

Figure 5 shows the format of a *data packet*. Data packets typically follow IN, OUT, and SETUP token packets. The originator of the data packet is the host for an OUT or SETUP token, or the device for an IN token.

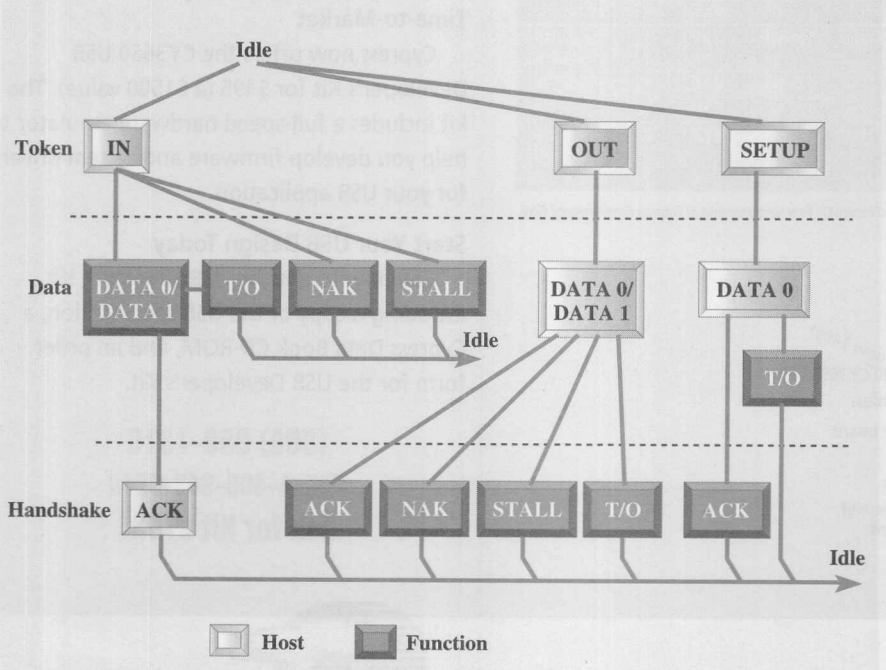
There are only two packet identifiers defined for data packets, DATA0 and DATA1. They are designed to support toggle synchronization when transmitting multiple data packets. Data packets must be an integral number of bytes long, to a maximum of 1,023 bytes. Note that the data packet has a 16-bit CRC.

*Handshake packets* simply consist of an 8-bit PID, with no CRC. Permissible values for a Handshake packet include ACK, NAK, and STALL. A NAK indicates that an endpoint is currently unable to accept data from the host or that it currently has no data to transmit. Note that a host can never issue a NAK; it must always be prepared to send and receive data. A STALL Handshake packet indicates that a function has serious problems and that some host intervention is required to clear the stall condition. This is done by sending a SETUP transaction to clear the stall. Note that SETUP token transactions can never be ignored—they must be accepted and acted upon by a device.

Currently, the only defined *special packet* has a packet ID of PRE. PRE packets signal a hub that a low-speed (1.5Mbps) transaction is about to take place and that any hubs with low speed devices attached should enable their downstream ports.

Figure 6 puts all these packet types together to show all the possible combinations. In reality it is a fairly simple and straightforward implementation.

**FIGURE 6**
*Complete USB transactions.*

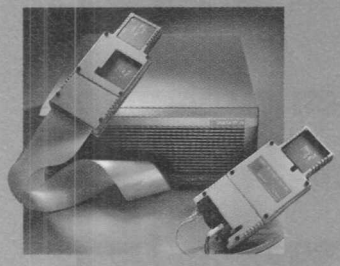



## TRANSACTION TYPES

There are four types of USB transactions. Each endpoint is defined to support one, and only one, transaction type. Transactions may be either bulk, control, interrupt, or isochronous.

*Bulk transactions* guarantee error free delivery of data between the host and a function endpoint. Bulk transactions start with the host issuing either an IN or OUT token as shown in Figure 6. If the host is writing data, it issues an OUT token followed by a DATA0 packet. The function will then respond with an ACK, NAK, or STALL. The ACK signifies that the data was received correctly and the host may send the next packet, if any, in the data sequence. Note that the next packet in the sequence would be transmitted using a DATA1 PID for the data transaction. If the data packet contains a CRC error, no handshake is transmitted by the function and a time-out occurs.

# *Now.* One tool gives you all the 80960 debugging punch you need.

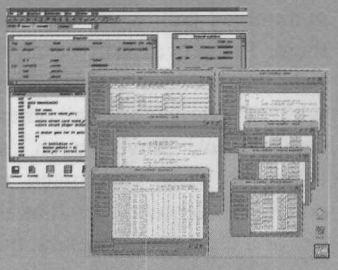


**Precise** *RP debugging, complete control and execution trace, plus high- and low-level views of PCI bus activity*

**New** *PCI trace shows time correlation between PCI bus and CPU bus*

**Flexible** *tool set offers full-featured CodeICE™ and affordably priced CodeTAP® emulators*

**Easy** *graphical debugger and disassembler for Sun, HP 9000 and Windows makes it easy to put bugs on the mat*



**Powerful** *CPU Browser™ lets you see graphical views of register states and modify them on the fly*

# TOTAL IMPACT

**Agile** *RTOS-Link™ allows integrated operating system debugging*

**Ask us about our support for:**

*80960Hx series*
*80960Jx series*
*80960CA, CF*
*80386EX*
*80C186EA, EB, EC*

## CALL NOW 1-800-426-3925

for your free information packet • Browse http://www.amc.com • E-Mail: info@amc.com

Applied is a supporting member of the Intelligent I/O SIG.

Applied Microsystems Corporation

European Distributor: Applied Microsystems UK and Europe, Tel: +44(0)1296-625462, Fax: +44(0)1298-623460; Germany Tel: +49(0)89-427-4030, Fax: +49(0)89-427-40333



**CIRCLE # 20 ON READER SERVICE CARD**

When a time-out occurs, the host will retransmit the data packet using the same PID as the failed packet.

Bulk transactions do not have any guaranteed bandwidth associated with them. In fact, as will be shown later, bulk transactions use whatever bandwidth is left over after all other types of transactions have been serviced. However, when guaranteed, error-free delivery is required, such as in a scanner or a digital still camera, bulk transactions are the way to go.

*Control transfers* are used for short command/control messages between a host and a function. There are typically two stages in control transfers, setup and status. There may also be a data stage in between the setup and status stages. As mentioned previously, functions cannot ignore setup tokens; therefore, a `NAK` or `STALL` status response is not permissible. If the `SETUP` token packet is corrupted, it should be ignored and a time-out will occur.

**FIGURE 7**
*Control transfers.*

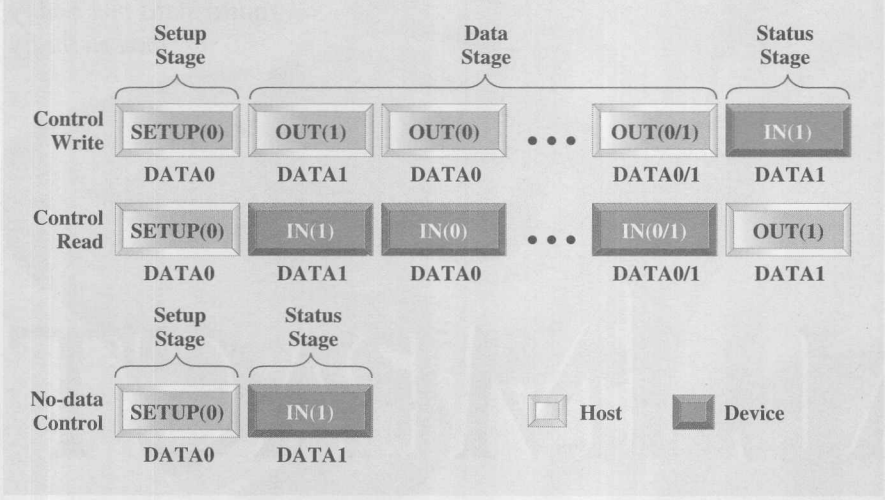
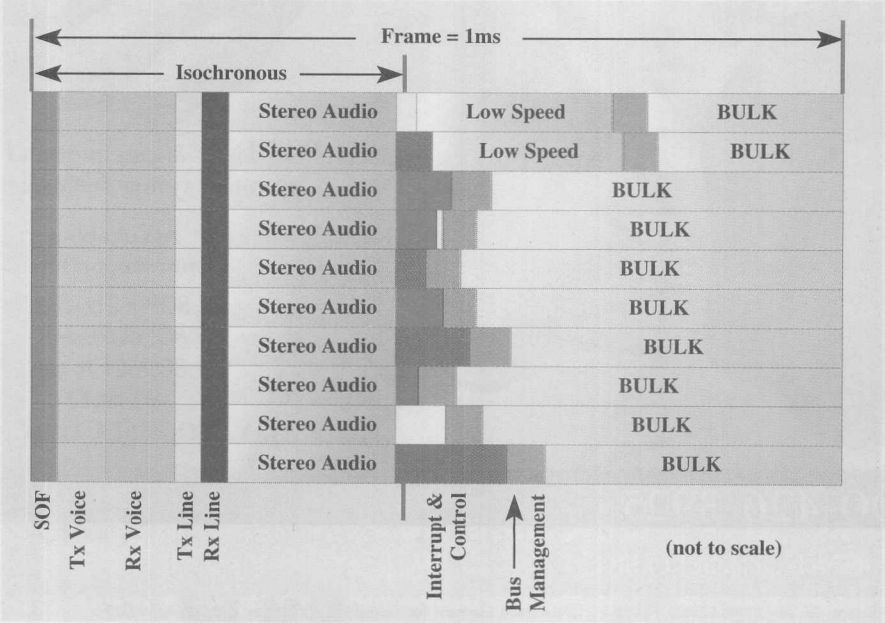



**FIGURE 8**
*USB frame structure.*




**When guaranteed, error-free delivery is required, bulk transactions are the way to go.**

All control transfers start with a setup stage. If there is a data stage to follow, it will adhere to the same rules as a bulk transfer stage. Finally, the status stage completes the handshaking sequence. Figure 7 illustrates the three possible types of control transfers.

*Interrupt transactions* are a bit of a misnomer. No device other than a host can initiate a USB transaction. What a device can do, however, is have the host make a regularly scheduled polls of the device. The frequency of polling is specified by the device during the bus enumeration process, which is described later. The polling process simply consists of the host sending out an `IN` token packet to the device and the device responding with either a data packet (if new data is available) or a `NAK` or `STALL` (if data is not available). Note that the maximum allowable data payload for an interrupt transaction is 64 bytes.

One of the key elements of USB for telephony and audio applications is the ability to provide a guaranteed amount of bandwidth for a particular peripheral. USB uses *isochronous transactions* to provide a constant bandwidth, error tolerant data transfer mechanism. A device that requires a certain throughput rate will request it during the discovery and enumeration process. During each USB data frame, which occur at 1ms intervals, the host will either read from or write to the isochronous device, depending on the direction specified by the device at enumeration time. This process is similar to the interrupt transactions but with a few significant differences. First, the

**FIGURE 9**
*USB communication layers.*

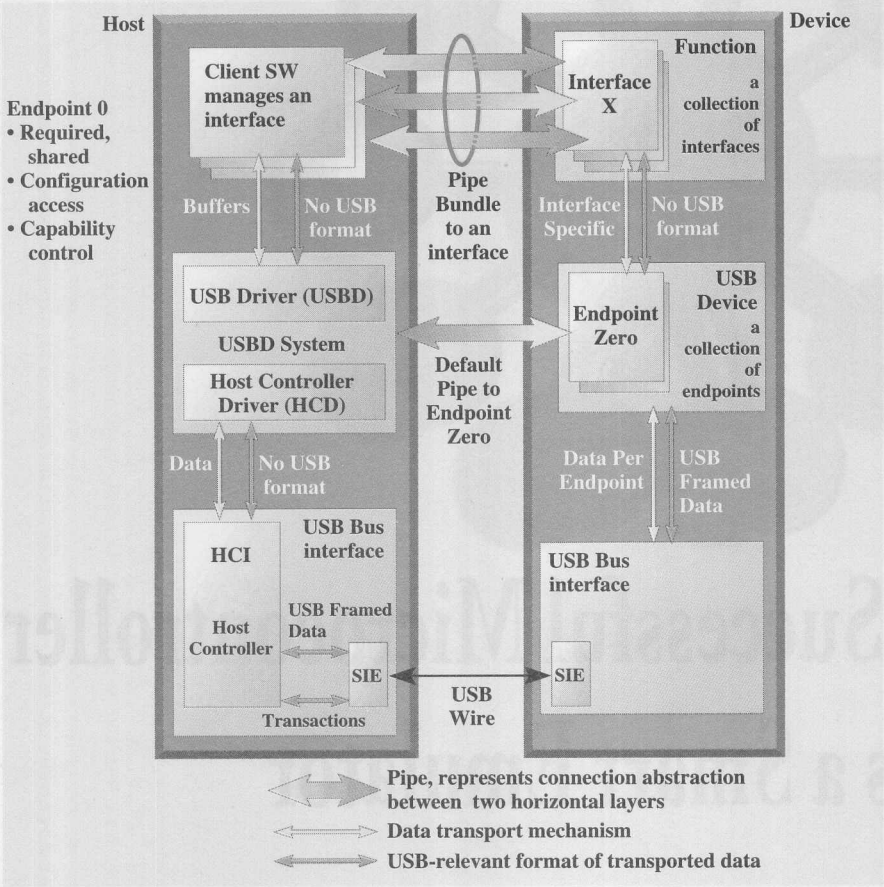
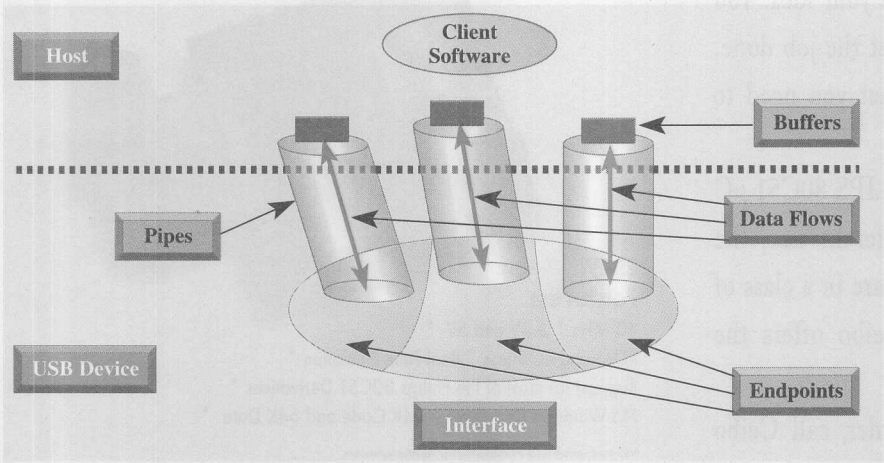



**FIGURE 10**
*USB device software entities.*




isochronous transfers occur every frame, while the interrupt transfers may or may not, depending on the requested interrupt frequency. Second, the isochronous transfers may be up to 1,023 bytes long as opposed to the 64 byte limit of an interrupt transfer. Finally, and most importantly, isochronous transfers are not acknowledged. For this reason, the data must be error tolerant. A CRC is present, so errors may be detected, but there is no capacity for re-transmitting erroneous data. If error-free transmission of large amounts of data is required, such as in the case of a scanner or digital still camera, bulk transactions should be used. If guaranteed bandwidth is of the utmost importance, such as in audio applications, then isochronous transfers are the way to go.

## USB FRAMES

The USB host controller breaks up data transfers into frames, which occur at 1ms intervals. During each frame, time is allotted for each transaction type with the following priority:

- Isochronous transfers
- Interrupt transfers
- Control transfers
- Bulk transfers

Note that it is possible to have devices on the bus requesting more bandwidth than is available. In this case, the host would have to notify the user of this fact. The end user would need to remedy the situation, most likely by removing a device. Some devices could also have the capability to operate in a lower speed mode, such as a video camera operating at 10 video frames/sec instead of the normal 30 video frames/sec. This situation might free up enough bandwidth to allow all the devices to remain on the network.

All frames begin with a start-of-frame (SOF) packet, which includes the 11-bit frame number, and end with an end-of-frame (EOF) idle interval. After the SOF packet, the host services all isochronous devices on the network. Once isochronous transfers are complete, interrupt, control and bulk transfers are handled in succession. Figure 8 shows an example of a network with five isochronous functions: voice transmit and receive, modem data transmit and receive, and stereo audio. Note that the isochronous portion of each frame is always the same length, hence the guaranteed bandwidth.

# POWER³

2500ADSOFTWARE

iSYSTEM
THE TOOL COMPANY

Avocet Systems announces the aquisition of 2500AD Software. Avocet and 2500AD each have 18 years of experience, coupled with iSYSTEM's 10 years of In-Circuit Emulator development. Avocet now offers professional Compilers, Assemblers, Simulators, Real-Time Operating Systems, and In-Circuit-Emulators for more embedded targets than any other company. Now you can get all of these tools from one source with one contact for technical support.

IT'S AVOCET'S COMPLETE PACKAGE.
It's your complete solution.

AVOCET

http://www.midcoast.com/~avocet avocet@midcoast.com 800-448-8500 207-236-9055 Fax: 207-236-6713
PO Box 490, 120 Union St., Rockport, ME 04856

CIRCLE # 22 ON READER SERVICE CARD

## USB DATA FLOW

The preceding sections of this paper described the physical packet level side of USB. From the logical data flow perspective, there is much more to it than that. There are several layers of software in the USB model.

On the host side, there is the host controller driver (HCD). This layer abstracts the actual USB controller hardware from the USB system. The upper layers need not know whether this layer is talking to a USB controller add-in card or to a USB controller built in to the computer's main chipset. The USB system software communicates using the host controller interface (HCI). Currently there are two HCIs, the universal host controller interface (UHCI), which is provided by Intel and supports their chipsets, and the open host controller interface (OHCI), which was developed by Compaq, Microsoft, and National Semiconductor. While it may be a bit disconcerting to have two competing standards, developers of USB peripherals and host applications will typically not come in contact with the HCI layers; contact will be handled by the operating system transparently.

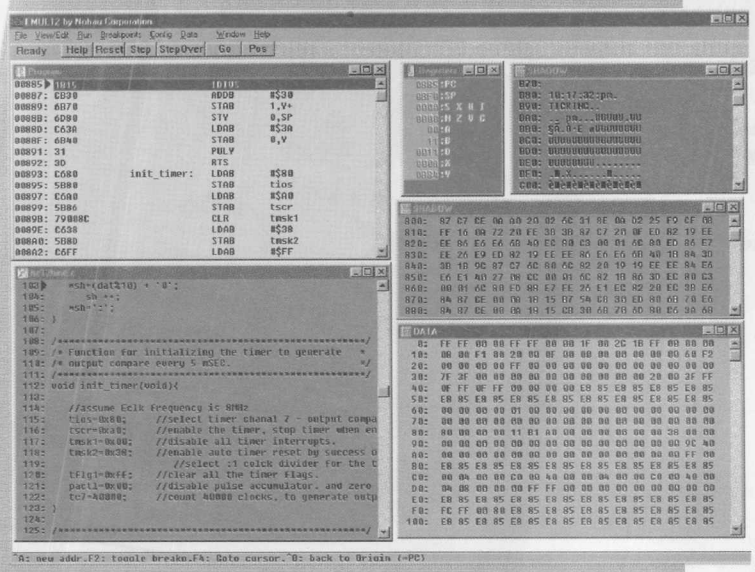
Above the HCI sits the USB system software. This software manages the USB resources, such as allotting bandwidth, assigning addresses, scheduling interrupt transaction service, and so on. The system software communicates to the host controller via the USB driver as do the client side applications. One note of caution to client side application developers: Microsoft's support of USB rests on the delivery of their new Win32 Driver Model (WDM) for Windows 95, the forthcoming OS97, and Windows NT. The WDM will look very similar to the current Windows NT drivers with differences in the driver setup and shutdown procedures. Microsoft has been somewhat tardy in delivering the WDM, and its associated generic class drivers, to OEM manufacturers, which has caused a significant delay in the proliferation of USB devices into the market.

USB devices also have logical layers of functionality. Each of those layers and some USB terminology will be discussed subsequently.

## LOGICAL LAYERS

*Endpoints* are the sources and sinks of all USB communication flow between the host and a peripheral. Endpoints have several characteristics associated with them, including the transfer type, bandwidth requirements (if isochronous service is specified), interrupt frequency (if interrupt service is specified), maximum packet size, and transfer direction of data (if bulk or isochronous service is specified). Note that each endpoint can only have one transfer type and one

# 68HC12 In-Circuit Emulators




## Real-Time Microprocessor Development Systems

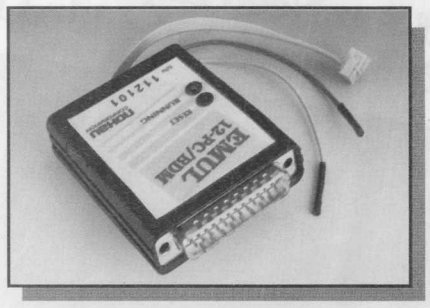
## EMUL12™-PC/BDM

### EMUL12-PC/BDM FEATURES:

- Microsoft *Windows* user interface.
- Hosted on PC's or workstations.
- Connects to PC through standard parallel port or through plug-in board.
- Real-time emulation at maximum chip speed.
- Single-step in RAM and ROM.
- Programs $E^2$ and Flash.
- Software breakpoint placement "on-the-fly".
- In-depth high-level support for popular C-compilers.
- Supports targets working with 2.5V to 5V power supply.
- Access to internal and external memory while emulating.

MICROSOFT WINDOWS/'95 COMPATIBLE

***To learn more,*** *call* ***(408) 866-1820*** *for a product brochure and a FREE Demo Disk. The Demo can also be downloaded from our web site-* ***http://www.nohau.com***



*For more information via your Fax, call our 24-hour Fax Center at (408) 378-2912.*


nohau CORPORATION

51 E. Campbell Avenue
Campbell, CA 95008-2053
Fax. (408) 378-7869
**Tel. (408) 866-1820**
Email: sales@nohau.com
*web: http://www.nohau.com*


**Support also available for:**

**8051 P51XA™ MCS®251 80C196 MCS®296 68HC11 68HC16 683xx C166**


Argentina 54 1 312-1079/9103 , Australia (02) 654 1873, Austria 0222 277 20-0, Benelux (078) 681 61 33
Brazil (011) 453-5588 , Canada 514-689-5889, China +86-10-2059495, Denmark 43 44 60 10
Finland 90-887 33 330 , France (1) 69 41 28 01, Germany 07043/40247 , Great Britain 01962-733 140
Greece +30-1-924 20 72, India (0212) 412164, Israel 03-6491202, Italy 02-49-82-051, Japan (03) 3405-0511

NOHAU INTERNATIONAL

Korea (02) 784-7841, New Zealand 09-3092464, Norway 22 67 40 20, Portugal 01 421 3141
Romania 056 200057, Singapore +65 749-0870, S.Africa (021) 234 943, Spain (93) 291 76 33
Sweden 040-59 22 00, Switzerland 01-745 18 18, Taiwan 02 7640215, Thailand (02) 668-5080

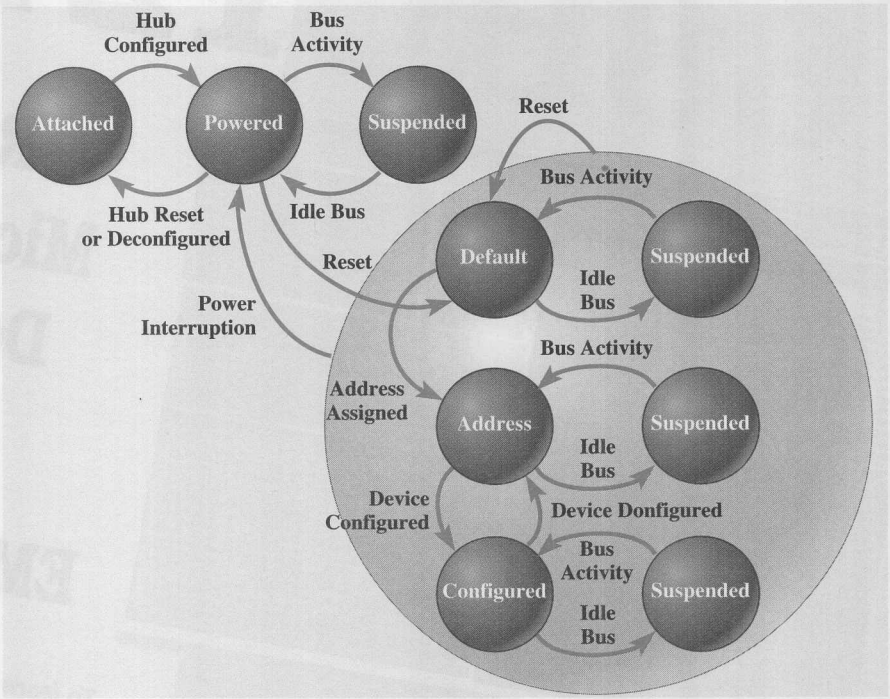

**CIRCLE # 24 ON READER SERVICE CARD**

direction associated with it. Full-speed USB devices may have up to 16 input endpoints and 16 output endpoints. Low-speed devices are limited to three total endpoints.

All USB devices must have at least one endpoint and that endpoint will have an endpoint number of zero. Endpoint 0 is used to initialize the device during the enumeration process and to provide configuration information about the device; therefore, endpoint 0 must always be defined as a control transfer endpoint. It may also be used after enumeration as a generic endpoint for control transfers.

*Pipes* are the logical channels formed between a device endpoint and the host's data buffers. Pipes are created when a device is configured, and they assume the characteristics of the endpoint with which they are associated. Two types of pipes are specified:

**FIGURE 11**
*Device state machine.*



# hitex DEVELOPMENT TOOLS

Genuine Real-Time Emulators!

## MORE Powerful New Products at Hitex Emulators that work !

### 8051 AX51 & MX51 Emulators

Hitex is Europe's leading 8051 emulator supplier. Well over 150 derivatives from manufacturers such as Intel, Dallas, Philips, Siemens, Atmel and AMD are supported. Recent additions include the Siemens 515C (CAN interface) and 504L series and the Intel 87C51FC... with more to come soon. Hitex probes are the smallest in the industry and each probe can support many variants lowering your costs and maximizing the utility of your Hitex equipment. Hitex has both bondout and standard chip Emulators offering an unbiased selection to you. Hitex 8051 emulators are powerful with features that speed your projects to completion. For a list of these Hitex Competitive Advantages, see our Web site or phone us today. Hitex Hardware Breakpoints stop the emulation before executing the instruction where the breakpoint is set. A Complex Trigger system coupled with a filterable Trace makes this a most powerful emulator.

### MCS®251 AX251 Emulator

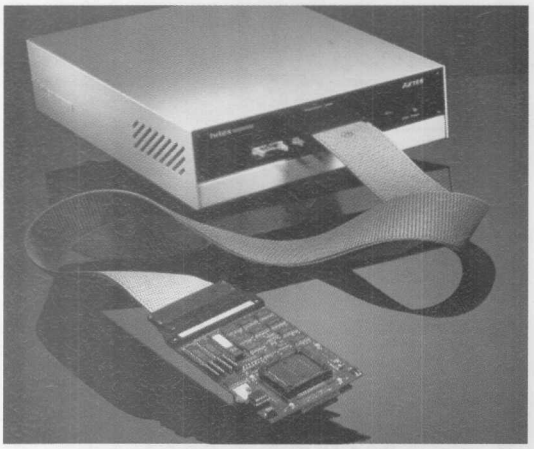
The AX251 is the latest 16 bit emulator design from Hitex for the Intel MCS®251 microcontroller family. Trace memory and all software is included in the purchase price. The Intel 8x251SB microcontroller is supported and future variants will require only a pod change or a software driver. The Universal Serial Bus (USB) chips (8x930AX & 8x930HX) are also supported increasing the versatility of the AX251. The AX251 operates in genuine Real-Time and does not require any resources from the target system which facilitates effective debugging sessions for the designer. The advanced AX251 is shipping from stock today and technical support is based in California. Hitex products are sold world-wide. The hardware breakpoint system consists of separate 256K execution and 256K data access breakpoints. You can set a breakpoint on a variable name and an address. Breakpoints stop the emulation before executing the instruction where the breakpoint is set.

### Siemens C161 AX161

Siemens introduces a new low cost processor - the C161 - Hitex is now pleased to announce the new low cost AX161 dedicated C161 In-Circuit Emulator with a full-featured bundled Keil 161 C Compiler. This emulator is shipping now - when needed. It utilizes Hitex standard chip emulation techniques eliminating version problems. The AX161 is functionally equivalent to the popular AX166 emulators. At a price low enough to please everyone ..... It has 20 hardware breakpoints, 2 Complex Trigger systems, 4 K frames or 2048 C Lines trace buffer, 256 Kbytes 0 WS memory and a Real Timer. Genuine Real-Time operation - no cycle stealing. Emulator functions can be accessed while the emulation is running without stealing target cycles. The AX161 uses same friendly HiTOP user-interface as all other Hitex products. This unit will have upgrade capabilities to Bondout technology and other derivatives.

### Siemens C167 AX166

Hitex offers the widest range of emulators available for the entire Siemens 166 family including the 166, 167, 167CR, 165, the 163 and now the just announced 161. Both Bond-out and regular series chip emulation are offered to provide you with the emulator you really need. These are supported by the full feature teletest 32 and the economical AX series. The AX is now available in a bond-out version and is shipping NOW! The Hitex Bondout emulators can trigger on internal register values in Real-Time! This is an exclusive Hitex feature. Various derivatives are supported by changing the pod, a personality module or merely the chip on the pod. Your Hitex emulator will not become obsolete with the introduction of new derivatives. These emulators operate in genuine real-time at full processor speeds for powerful emulation. Clock cycles are never stolen due to the distributed processing in the emulator.



AX166/161 In-Circuit Emulator - Standard Chip

### CAN Network CANalyzer

Hitex distributes the Vector CANalyzer - the world's standard for CAN (Controller Area Network) simulation, testing and debugging equipment. The CAN bus is becoming popular for automotive and factory automation applications in North America. The CANalyzer can generate messages, gather statistical network information, simulate virtual nodes and reply to specified network messages. A intuitive user-interface provides a powerful point & click environment for fast setup and use of the analyzer. It supports both 11 and 29 bit identifiers. The CANalyzer comes in a ISA bus card and also a PCMCIA card for laptops. The software is Windows or DOS. Hitex also offers a 8 bit ISA card for user applications. The **HiCAN** card is accessed using standard IO space and contains an on-board processor to ensure traffic is handled without loss or corruption of messages. See our Web site or free CD-ROM for general CAN info.

**Hitex Development Tools**
**2055 Gateway Place #400**
**San Jose, CA 95110**

☎ (800) - 454-4839
☎ (408) - 298-9077
Fax: (408) 441-9486
email:info@hitex.com
http://www.hitex.com

Hitex also supports:

| | |
|---|---|
| 8051 | 68HC11 |
| 683xx | 386EX/DX |
| 186EA/EB/EC/EM/ES/ER | |

- *Stream*: data that has no structure relating to the USB spec, such as an audio stream
- *Message*: data that has a defined USB structure, such as a configuration command

Note that because all devices have an endpoint 0, all devices have at least one pipe, which is called the default pipe. As mentioned previously, the default pipe is used for all configuration functions and may be used subsequently for other generic control transfers, although "ownership" of the default pipe is always held by the host's USB system software.

Using the WDM model, the client-side software will typically request a data transfer via an I/O request packet (IRP) to a pipe. The client will then block on the IRP and wait, or be notified via an interrupt.

An *interface* is a collection of endpoints that, taken together, provide the means to interact with a specific function. A function is defined as something that adds capability to a host, such as a printer or fax machine. From the client viewpoint, function interfaces are the entities bound to device drivers. Consequently, the pipes that are associated with the endpoints comprising the function interface are the means of controlling and transferring data between the client and the function.

Note that, during configuration, alternate interfaces may be selected for a specific function. An example of this might be a digital camera that can support the generic imaging class drivers that allow it to transfer images to the computer. During configuration, if only the generic driver is installed on the host machine, the configuration process would select the default function interface. If the end user has a vendor supplied driver that supports looking at thumbnails of all the images stored in the camera or other enhancements, the configuration process could select the alternate function interface that supports these capabilities.

Functions must have at least one endpoint. In fact, all functions must share endpoint 0. In addition, high speed devices may have an additional 15 `IN` and 15 `OUT` endpoints.

*Devices* are the highest layer of abstraction in the USB model. Logical devices can be either hubs or a collection of one or more functions. Each device is accessed by a unique USB address that is assigned by the host during the enumeration process. The logical device is the container that holds all the interfaces and characteristics that enable the peripheral to participate successfully in USB's plug-and-play environment.

**FIGURE 12**
*USB enumeration states.*

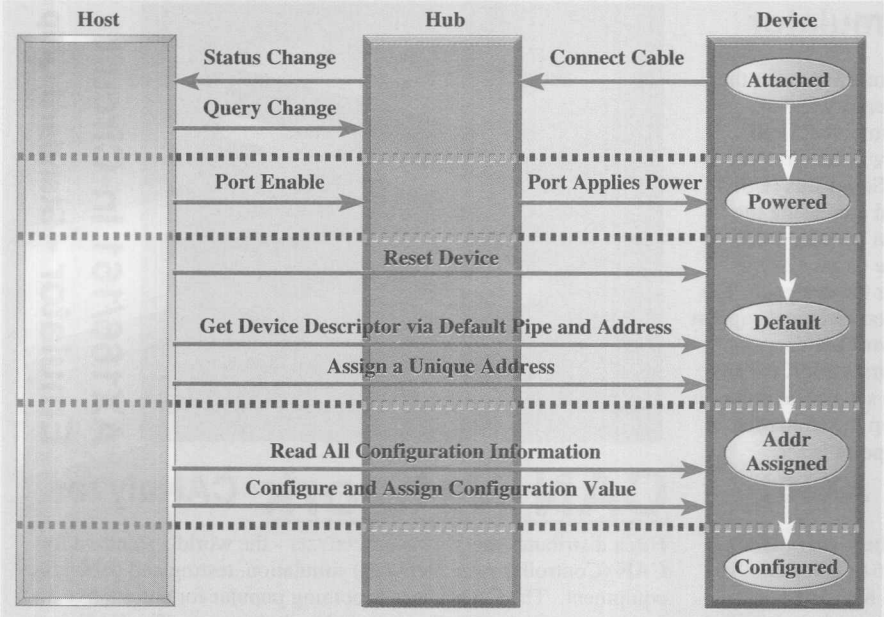



**FIGURE 13**
*USB device request format.*

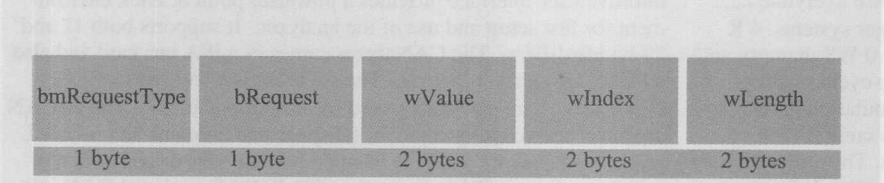
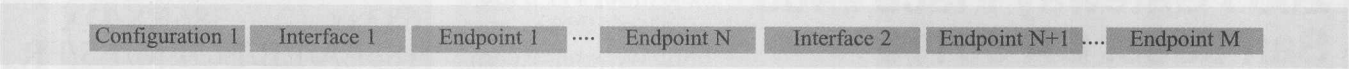


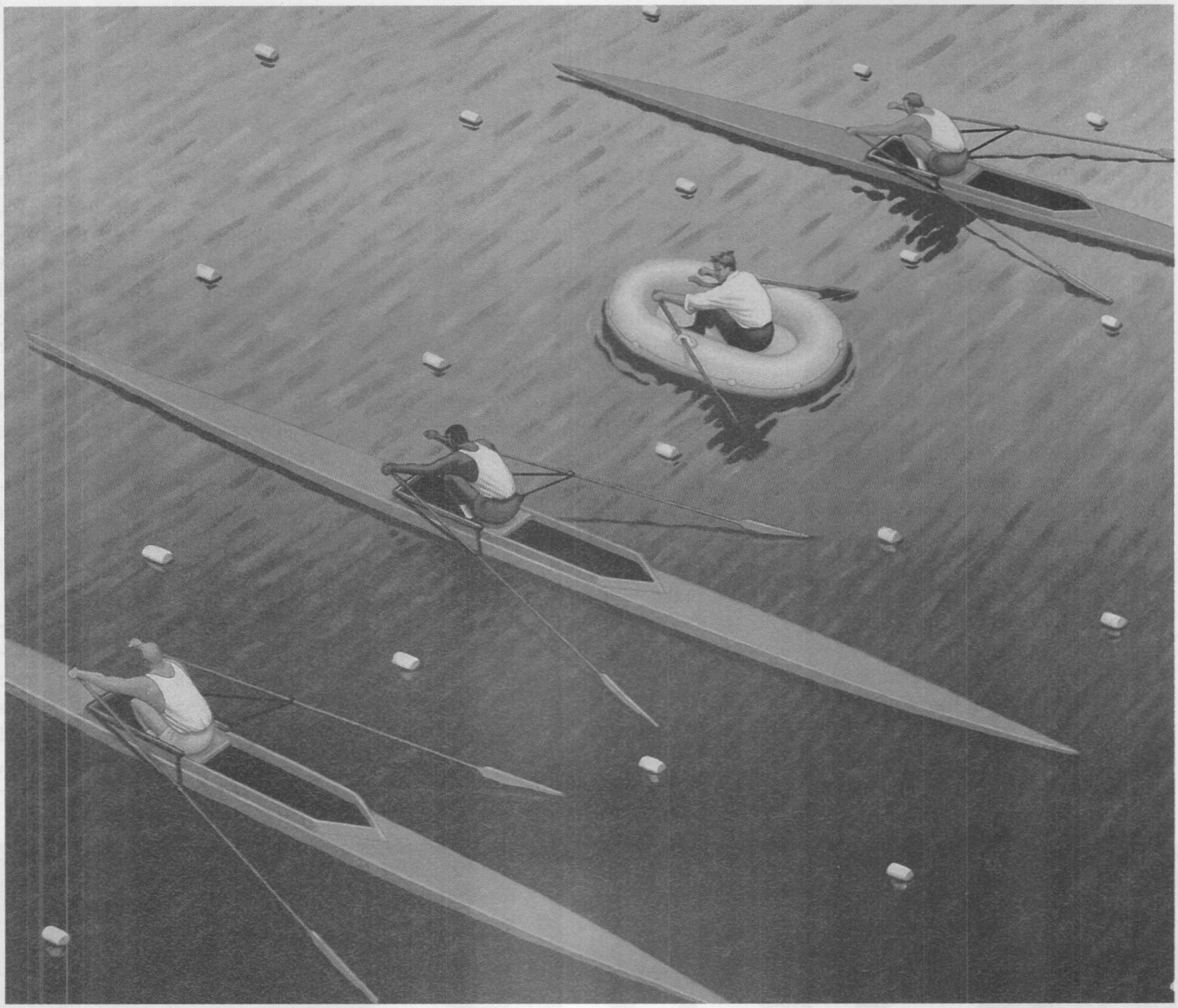

**FIGURE 14**
`GET_CONFIGURATION` *data response.*




## DEVICE FRAMEWORK

The USB device framework consists of the items in the following list:

- USB operational state machine
- Common set of generic USB operations
- Common set of device requests generated by the host that must be responded to



# It's hard to compete without the right tools.

**Nobody can pinpoint problems in a Pentium® processor or Pentium® Pro system like we can.**

American Arium is the undisputed leader in providing in-circuit emulators to PC manufacturers and BIOS developers. These tools and our 15+ years of ICE experience are now available to you.

Using WinDb™, our Windows™-based interface, you can:

- **View source, assembly and trace bus cycle data stored at the full 66 MHz bus speed**
- **Create multi-level complex triggers**
- **Stop the processor on memory, code and I/O accesses and other conditions**
- **Probe 16 external channels**
- **Create automated tests and eliminate redundant steps with our C-like command language**
- **Call or e-mail for free technical support**

If you can't afford to finish in second place, visit our website or call us today. Ask how your project can benefit using the right tools for the Pentium® processor and Pentium® Pro.

american arium

14281 Chambers Road, Tustin, CA 92780
(714) 731-1661 • FAX (714) 731-6344
www.arium.com

Pentium and Pentium Pro are registered trademarks of Intel Corporation.
Windows is a trademark of Microsoft Corporation. WinDb is a trademark of American Arium.

**CIRCLE # 102 ON READER SERVICE CARD**

- Set of standard device descriptors
- Optional set of class- and/or vendor-specific descriptors

### THE USB STATE MACHINE

A USB device has many possible states. However, not all of these states are visible to the firmware resident on the peripheral. Some of them are transient states controlled by the USB controller, be it an external chip, an off-the-shelf microcontroller with an integrated USB port, or a custom ASIC with a USB core. The state machine is shown in Figure 11.

The states that are visible to the upper level software are as follows:

- *Attached:* the device is attached to a USB network, but the interface has not been powered
- *Powered:* the device is attached to a USB network and power is being applied to the interface via a hub. The interface has not yet been reset. USB devices may be powered by the bus or externally (self-) powered. Note that even self-powered devices must report the maximum amount of power they will draw from the USB bus. Bus powered devices that draw more than one USB load (100mA) must be limited to drawing only one USB unit load until after the device has been configured. At that time a device can draw up to five unit loads, with the actual maximum level being reported in the configuration information of the device
- *Default*: the device is attached, powered and has been reset. It has not yet been assigned a unique address, only responds to the default address (0x00), and uses the default pipe
- *Address assigned:* the device is attached, powered, has been reset, and has been assigned a unique 7-bit address. No configuration has yet taken place. The device still only responds to requests on the default pipe, but now only at the assigned address
- *Configured:* the device has reached the address assigned state and has been configured by the host. The configuration process will be discussed subsequently. Note that at this time, the device can be used by the host to provide its intended function
- *Suspended:* as a power saving measure, USB devices must automatically enter the suspended state when the device has seen no bus traffic for a specified period of time. When a device is suspended, it retains any address and configu-


# *RTI* has the tools you *need* for TORNADO™

**StethoScope®**
**ControlShell™**
**NDDS™**
**ScopeProfile™**
**RTILib™**

- Application Visualization and Debugging
- Visual Programming for Electromechanical Systems
- Performance Measurement and Tuning
- Real-Time Network Connectivity to Windows® & Unix
- Memory Analysis and Leak Detection

***Shaping the Future of Real-Time®***

***Maximize* your Tornado investment.**
**Call, email, or visit RTI's web site:**
**(408) 720-8312 • info@rti.com**
**http://www.rti.com**

***REAL-TIME INNOVATIONS, INC.***
*155A Moffett Park Drive, Suite 111*
*Sunnyvale, CA 94089*

rti

ration data. The suspended state is terminated as soon as some bus activity is detected

A typical enumeration scenario is shown in Figure 12. When a device is attached to a hub, the hub signals the host of a change in status on one of the hub's ports. The host then queries the hub as to the nature of the change. When the host determines that a new device has been attached, it commands the hub to enable the port that the new device is connected to. At this point, the device changes from the attached to the powered state. This is the first time that the host can communicate directly with the device. From then on the sequence is fairly straightforward as the host resets the device, which puts it into the default state. Subsequently, the host uses the default pipe to assign an address and configure the device. Note that at any time after the device enters the powered state, if the device detects that the bus is idle for a specified period of time, the device will place itself into the suspended state. The device will exit the suspended state and return to the previous state as soon as any bus traffic is detected.

## GENERIC DEVICE PROPERTIES AND OPERATIONS

There are some specific properties and operations that all USB devices must support. They include the following:

- *Hot swapping:* all devices must be physically able to withstand being connected to and disconnected from the bus, with power applied, without damage or causing damage to other devices. When a device is attached to a USB port, it must follow the state machine previously discussed
- *Address assignment:* all devices must be capable of changing their addresses to those assigned by the host
- *Configuration:* device must be configured in order for the host to utilize the device's function. During configuration, the host requests specific descriptors from the device. These descriptors are structures that contain information regarding the capabilities of the device. The five types of descriptors are device, configuration, string, interface, and endpoint. These descriptors will be described in detail in a later section. For now, let's say that the host can, by reading the descriptors, choose a specific

configuration for the device that suits the needs of the device drivers that are installed in the system

- *Power budgeting:* all USB devices may draw a maximum of 100mA from the bus before they are configured. Once the configuration is complete, they may draw up to 500mA. The maximum current drawn from the bus is specified in the device descriptor. One note of caution here: just because a device can draw up to 500mA does not necessarily mean that a port can supply that much current. If that is the case, the end user should receive an error message from the OS stating the error, and hopefully suggesting an alternative, such as trying a different port
- *Remote wake-up:* remote wake-up is not a required function of a USB device. In fact, if it is included, there must also be the capability to disable the feature. With remote wake-up, a device is able to cause the host to resume from the suspended state. A good example of this is an idle computer that is connected to a USB compatible phone. If an incoming call is detected, the phone can signal the host computer to wake-up and handle the call.

## USB DEVICE REQUESTS

All devices must respond to device requests from the host that occur on the default pipe. The default pipe uses control transfer type. Every request sent by the host has eight bytes in the following format:

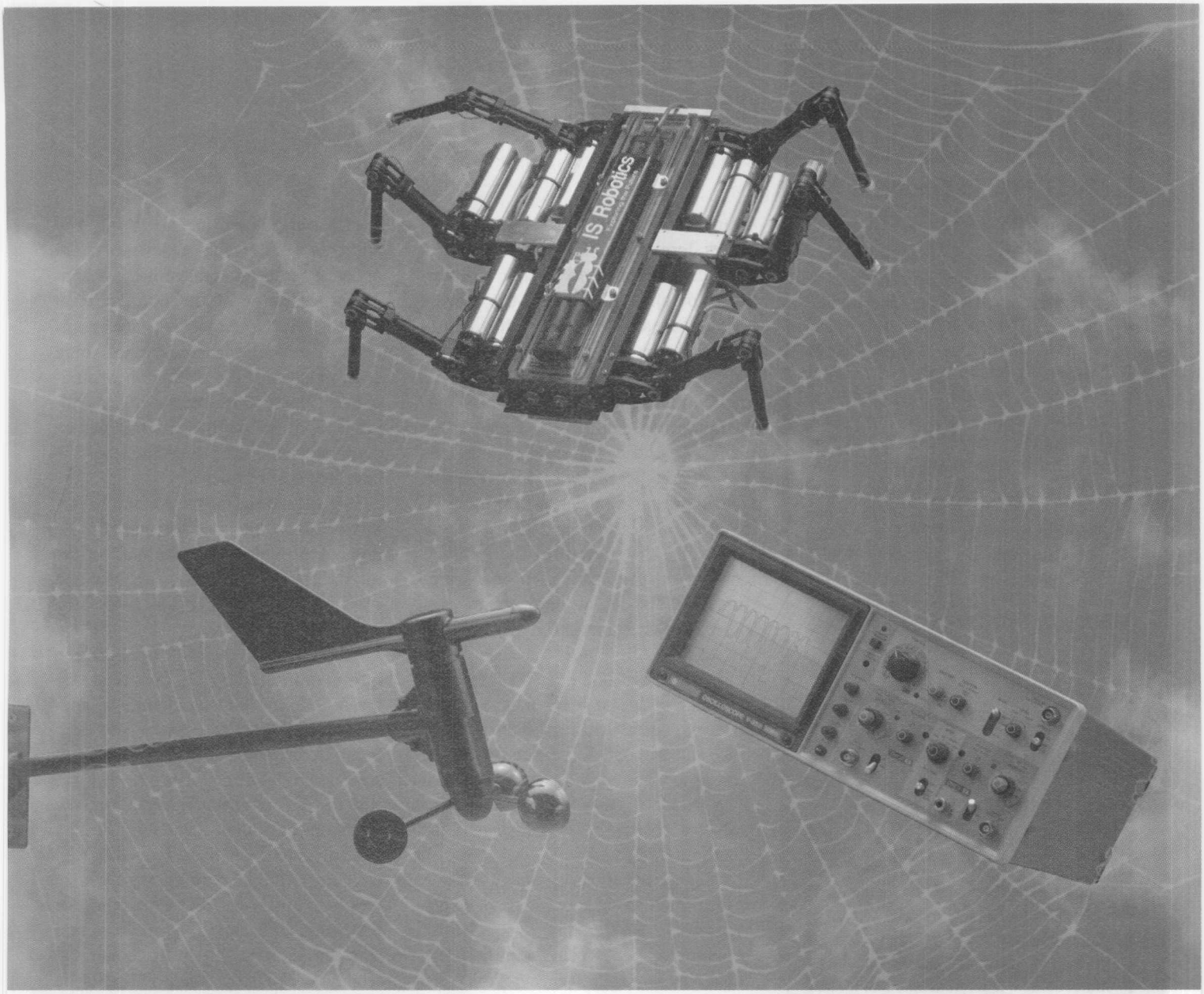
- `bmRequestType` is a bitfield with the following characteristics:
  - `D7` - Data transfer direction - 0 is host to device, 1 is device to host
  - `D[6:5]` - Request type - 0 = Standard, 1 = Class, 2 = Vendor, 3 = Reserved
  - `D[4:0]` - Request Recipient - 0 = Device, 1 = Interface, 2 = Endpoint, 3 = Other, 4-31 = Reserved
- `bRequest` is a byte whose meaning can vary depending on the request type field in `bmRequestType`. For a standard request type, the values can be `CLEAR_FEATURE`, `GET_CONFIGURATION`, `GET_DESCRIPTOR`, `GET_INTERFACE`, `GET_STATUS`, `SET_ADDRESS`, `SET_CONFIGURATION`, `SET_DESCRIPTOR`, `SET_FEATURE`, `SET_INTERFACE`, and `SYNCH_FRAME`
- `wValue` is a 16-bit word whose meaning depends on the value of the `bRequest` byte. For example, if the `bRequest` is `CLEAR_FEATURE`, then the `wValue` describes the feature that should be cleared. In the case of a standard request type, the `wValue` could be either `DEVICE_REMOTE_WAKEUP` or `ENDPOINT_STALL`. If the `bRequest` is a `GET_DESCRIPTOR`, the `wValue` would contain a descriptor type and descriptor index. Refer to table 9-2 in the USB Rev 1.0 specification for more information
- `wIndex` is a word whose meaning changes depending on the request type. Again, refer to Table 9-2 in the USB specification
- `wLength` specifies the amount of data to be transferred in the data phase of the control transaction. The transfer direction is specified in `bmRequestType, D7`. If the field is zero, there is no data phase

## DESCRIPTORS

Descriptors are data structures with a predefined format. All USB devices must report their characteristics and capabilities using descriptors. The five types of standard descriptors are device, configuration, string, interface and endpoint (as mentioned previously). There are also class and vendor specific descriptors. Both standard and class descriptors are essential to the plug-and-play process.

## STANDARD DESCRIPTORS

*Device descriptors* contain general information about the USB device and specific information about the default pipe. The device descriptor is a variable




# Don't Limit Your Product's Capabilities—Use Web Technology.

## Visit us at http://smallest.pharlap.com to see how you can develop embedded applications that take full advantage of the web.

**Making The Most Out Of Your Products**

Phar Lap's TNT Embedded ToolSuite,® Realtime Edition, complete with Realtime ETS™ Kernel, now comes with our robust networking protocol—ETS TCP/IP! This cutting-edge feature allows your customers' mainframes, workstations, or PCs to communicate with products on the factory floor, in the lab, or at remote sites—all using Web technology!

As a result, electronic OEMs can use this technology to create intelligent machines and instruments for an unlimited number of applications such as medical instruments, robotics, avionics equipment etc.

In addition, we support a variety of network protocols as well as industry standard tools such as Visual C++, Borland C++, CodeView, and Turbo Debugger.

**Never Underestimate The Power Of The Web**

Because Phar Lap's TNT Embedded ToolSuite comes with development tools, a Realtime ETS Kernel and ETS TCP/IP, it is a one-stop shopping product with enormous power!

To find out more, catch us on the "World's Smallest Web Server" today at URL http://smallest.pharlap.com or call a Phar Lap sales representative. Let your products realize the power of Web technology!

The 32-Bit x86 Experts

PHAR LAP

Embedded Development — Simply on Target™

Phar Lap Software, Inc. 60 Aberdeen Avenue, Cambridge, MA 02138 • Tel: (617) 661-1510 • Fax: (617) 876-2972 • http://www.pharlap.com

CIRCLE # 32 ON READER SERVICE CARD

length structure that can be up to 256 bytes long (although larger descriptors can be accessed using a proxy descriptor). Included in the device descriptor are the descriptor length and type (device), the USB revision to which the product was designed; the device class, subclass, and class-specific protocol supported, if any; some vendor-specific information such as the vendor ID, product ID, an index to a string descriptor describing the serial number, an index to a string descriptor that describes the product, and so on; the number of configurations that the device supports (at least one); and the maximum packet size for endpoint 0.

Even though one or more interfaces may use endpoint 0 for control transfers, its packet size is only defined once and that definition is in the device descriptor. Note that because endpoint 0 is standard among all devices, the only variable associated with it is the maximum packet size.

*Configuration descriptors* contain information about specific device configurations. Remember, as mentioned previously, a device can have more than one configuration available, but only one can be active at a time. Other items the descriptor describes are the number of interfaces supported by this configuration, the configuration value (which is used by the `SET_CONFIGURATION` request), the maximum power drawn by the device when this configuration is selected, and some configuration-specific attributes, such as whether the device is self-powered or remote-powered, and if it supports remote wake-up.

*Interface descriptors* directly follow their associated configuration descriptor in the data returned by a `GET_CONFIGURATION` request. Interface descriptors cannot be accessed independently of the configuration descriptors. Each interface descriptor is followed by its endpoint descriptors. If a configuration has more than one interface, the second interface descriptor follows the last endpoint descriptor of the previous interface as shown in Figure 14. Any additional interface descriptors are appended after the end of the last endpoint descriptor of the interface that precedes it. (Note that interfaces common to a configuration cannot share endpoints, with the exception of endpoint 0. Also, if an interface uses endpoint 0, it is defined in the device descriptor and not redefined in the interface descriptor.)

Interface descriptors contain information about the interface number, the number of endpoints contained in the interface, and any class, subclass and class protocol that might be associated with this interface.

*Endpoint descriptors* contain the information used by the host to set up the scheduling and bandwidth allocation desired by the device. The endpoint descriptors contain an endpoint address, which consists of a 4-bit address and one bit for direction; the endpoint attributes, (whether it is a control, isochronous, bulk, or interrupt endpoint); the maximum packet size the endpoint is capable of sending or receiving; and a polling interval that sets the interval (in milliseconds) at which an interrupt endpoint will be queried. Note that the polling interval must be set to one for isochronous endpoints and it is ignored for bulk and control endpoints.

*String descriptors* are unicode-encoded strings that contain human readable information about the device. String descriptors are optional, but if they are not supported, all references to them must be set to zero.

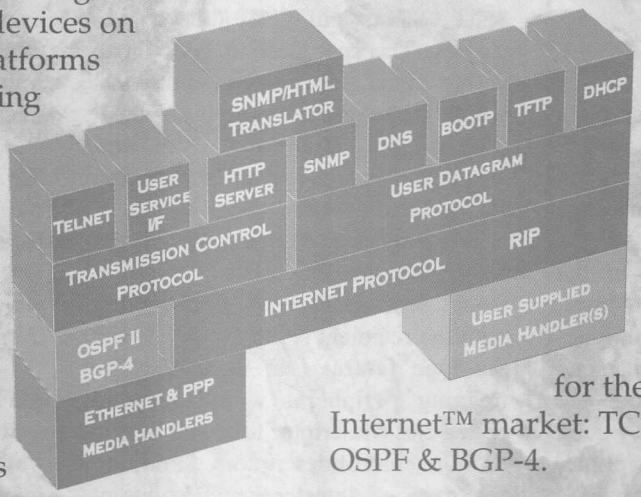
## CLASS DESCRIPTORS

Designing a product to belong to a specific USB class is entirely optional. If the product vendor wants to supply a proprietary driver without which the device will not operate, they may do so. If this is the case, it must be indicated by having the `DeviceClass` member of the device descriptor set to 0xFF. However, to truly participate in the plug-and-play configuration scenario, a product

# Q: What do pSOSystem™, Nucleus™, VxWorks™, Chorus™, & Your RTOS Have In Common?

# A: Networking Solutions From Epilogue.

With over 3 million embedded solutions deployed, Epilogue's Envoy™ SNMP is the most-proliferated embedded agent available. Our architecture allows designers to deploy managed devices on CISC and RISC platforms from 8 to 64 bits using today's most-popular embedded operating systems, or your own kernel - all from one ANSI C source code set.




Now, Epilogue has extended the power of SNMP by providing a seamless interface for your customers to use any on WWW browser to configure and manage devices with Decorum™.

Decorum allows OEMs to build a customized GUI tailored to your device, while providing all of the power and flexibility of standard SNMP management - the best of both worlds.

Complementing Decorum, Epilogue also offers a complete family of source code products for monitoring and connectivity solutions geared for the Embedded Internet™ market: TCP/IP, PPP, OSPF & BGP-4.

Find out what over 300 OEMs already know - Epilogue is your Source Solution for portable, compliant and supported network connectivity, monitoring and management.

EPILOGUE
Embedded Solutions
From Integrated Systems, Inc.

**www.epilogue.com**

**CIRCLE # 34 ON READER SERVICE CARD**

SuperComm
Come to booth 2162
to find out
more...

should belong to a specific device or interface class. With Microsoft's new Win32 Driver Model, for each USB device class there will be a generic class "minidriver" supplied with the OS that will support the basic functions of that class. If the manufacturers wish, they may supply their own drivers, which may make use of some added device features, but the devices should still operate without them.

The currently identified device classes are as follows:

- Audio
- Communication
- Hub
- Human Interface
- Image
- Monitor
- PC Legacy
- Physical
- Power
- Printer
- Storage

At this time, not all of these class specifications have reached the release stage. The most current information can be found on the USB home page (www.usb.org).

Many classes are defined at the interface level, not the device level. This situation allows for multifunctional devices to contain more than one class-specific interface without the need for implementing a hub. A good example might be a fax/scanner/printer device. The fax interface might belong to the communication class while the scanner would have an imaging class interface, and the printer functional interface would follow the printer class specification. Another issue with such a device is that the new multifunction peripheral interface (MFPI) communications protocol accounts for the multiple functions of the device. The designer must decide what class definitions to support. One solution to this particular question would be to support multiple configurations. One configuration could support the communications, imaging, and printer classes, as mentioned above, while the alternate configuration would allow the device to use the MFPI protocol. The host could then decide, based on what drivers are available, which configuration to select.

An example of a class specification is the imaging class. While, at the time of this writing, the imaging class is still very preliminary, it is a representative example of what a class specification contains. As mentioned previously, many classes are defined at the interface level, not at the device level, and the imaging class is no exception. When using an interface level class, the device descriptor's `DeviceClass` and `DeviceSubClass` both must be set to zero. Members of the imaging class would declare themselves as such in the `bInterfaceClass`. The `bInterfaceSubClass` entry further delineates the imaging class into `SCANNER`, `STILL_CAMERA`, and `VIDEO_CAMERA` subclasses. The specification goes on to define some specific endpoints that would be required by each subclass, such as a bulk `IN` for digital still cameras and scanners, and an Isochronous `IN` for video cameras.

When the host is preparing to configure an imaging class device, it would send a `GET_DESCRIPTOR` request with the recipient set to `INTERFACE` and the type (`wValue`) set to `CLASS`. The imaging peripheral would respond with a class descriptor followed by one or more control descriptors. Similar to the case of interface and endpoint descriptors, control descriptors would only be accessible by reading the class descriptor. Control descriptors would be used for setting such parameters as scanner resolution, exposure times, and so on. Controls could be implemented as :

- *Continuous types:* image size, scaling, cropping, exposure, and so on
- *Bitset controls:* start, stop, next index , previous index, and so on
- *Table-driven types*: such as a choice between 200, 300, 400 and 600 DPI on a scanner
- *Still-image modes:* such as the number of bits per pixel or the image format (JPEG, FlashPix, or T.6 Fax encoding, and so on)
- *Video image modes:* Raw RGB, CCIR-NTSC, CIF/QCIF, Indeo, and so on

These predefined control descriptors could then be used by either the generic imaging class driver or in an enhanced features mode by a driver supplied by the vendor. Since the imaging class is still preliminary, caveat emptor.

## NEXT TIME

In this article, we've looked at some of the details of the USB specification. For more detailed information, refer to the references listed below. Next time we'll look at some the USB peripheral controllers that are available and investigate some of the issues that need to be addressed when implementing a USB peripheral device. ESP

*John Canosa is a principal member of the technical staff at Questra Consulting, where he is responsible for designing and developing hardware and software for embedded designs.*

## REFERENCES

www.usb.org: *The USB Implementer's Forum Home Page*, the main source of information about USB, including the actual specification and pointers to USB products, controllers, cores, and so on.

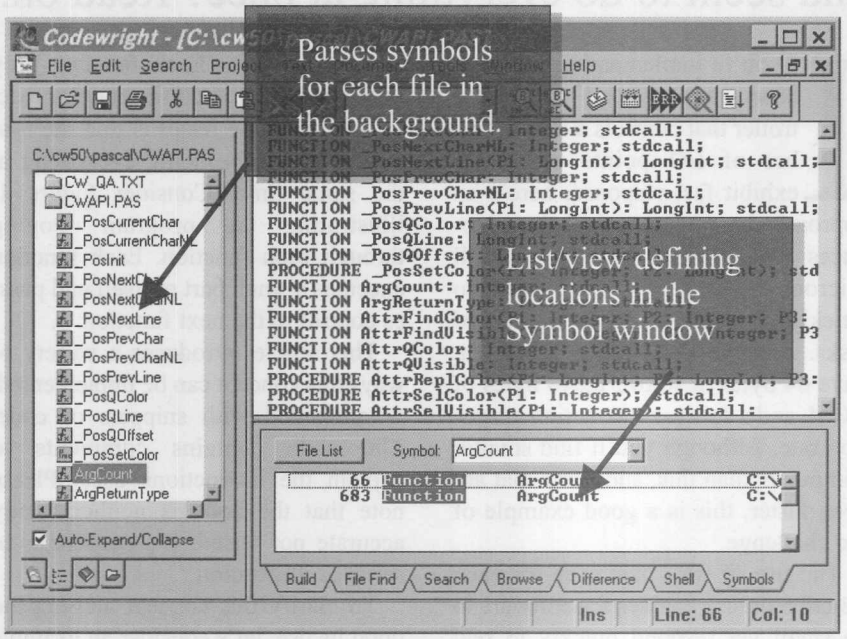
*Universal Serial Bus Specification, Rev. 1.0,* Hillsboro, OR: USBIF, 1996.

Anderson, Don, *USB System Architecture*, Reading, MA: Addison-Wesley, 1996.

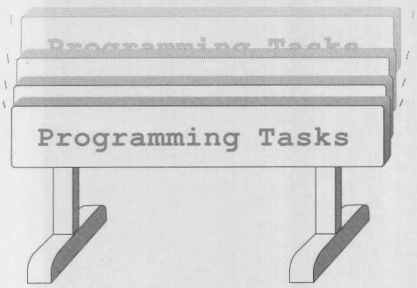
www.intel.com/embedded: Garney, John, "An Analysis of Throughput Characteristics of Universal Serial Bus", Media and Interconnect Technology, Intel Architecture Labs.

www.mindshare.com: USB Training Courses, Mindshare Inc. (719) 488-8990.

# Raise the bar
## by lowering your hurdles.




Navigate code *fast* with Outline Symbols




With a

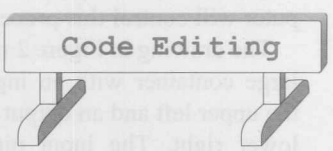
Codewright™ V5.0

Professional

Programmer's Editor

from premia®

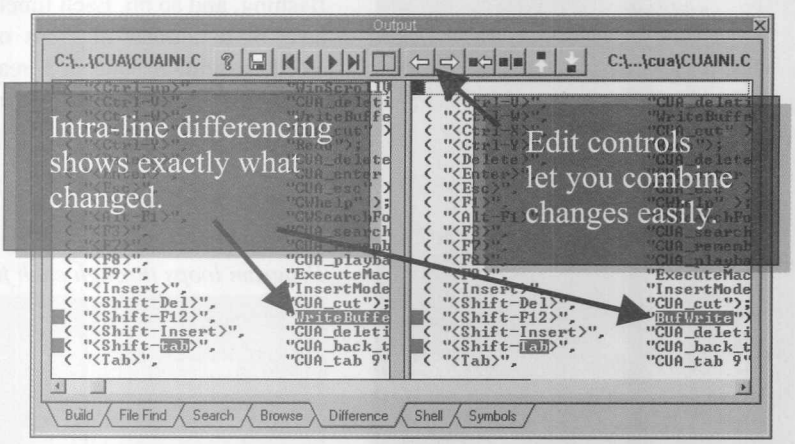
*Premium Quality Software*




Codewright helps you over tasks like:

- Locating and navigating to specific code.
- Creating and maintaining comments.
- Working with version control.
- Typing repetitive constructs.
- Renaming classes and variables across hundreds of files.
- Focusing on what's important.
- Working with others *and* their revisions.
- Linking to non-text files.
- Your next raise (maybe).

Codewright synchronizes with your Visual C++ and Delphi environments, and offers the best free support in the business. Call about a free evaluation.




Reconcile changes with Difference Editing

It's **Tool One.**

Call now: 1-888-4-PREMIA
477-3642

http://www.premia.com

Single user license:
Codewright Professional
for Windows, Windows 95, and NT.

**$269**

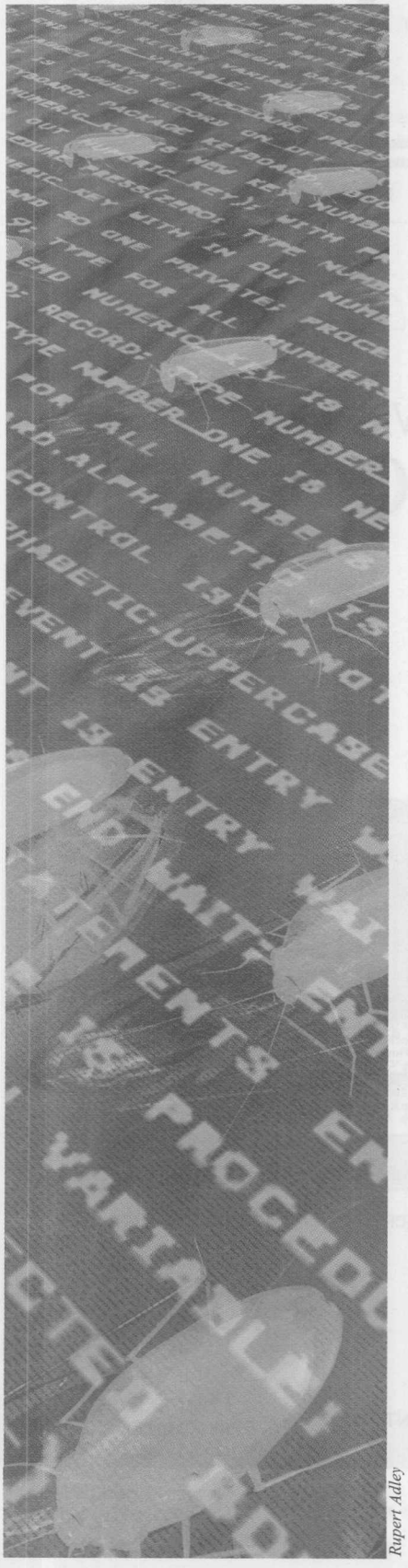
Volume discounts and site licenses available.

*Premia Corporation*
*1075 NW Murray Blvd., Suite 268*
*Portland, Oregon 97229 USA*
*Fax: +1-503-641-6001*
*Phone: +1-503-641-6000*
*Email: sales@premia.com*

**CIRCLE # 35 ON READER SERVICE CARD**

# State-Oriented Programming

by Al Schneider




Want to know how to allow small microcontrollers to be fast, use very little memory, and seem to do everything at once? Read on.

If you're implementing a system based on a small microcontroller that controls a large number of I/O points and must exhibit fast response, here's an approach to programming that produces fast, compact code for small microcontrollers, enabling them to quickly perform a wide variety of tasks. A small microcontroller might have 64 bytes of RAM, 2,048 bytes of ROM, and a 1$\mu$s per instruction execution rate. Although you'll find smaller computers than this, and some that are even faster, this is a good example of the challenge.

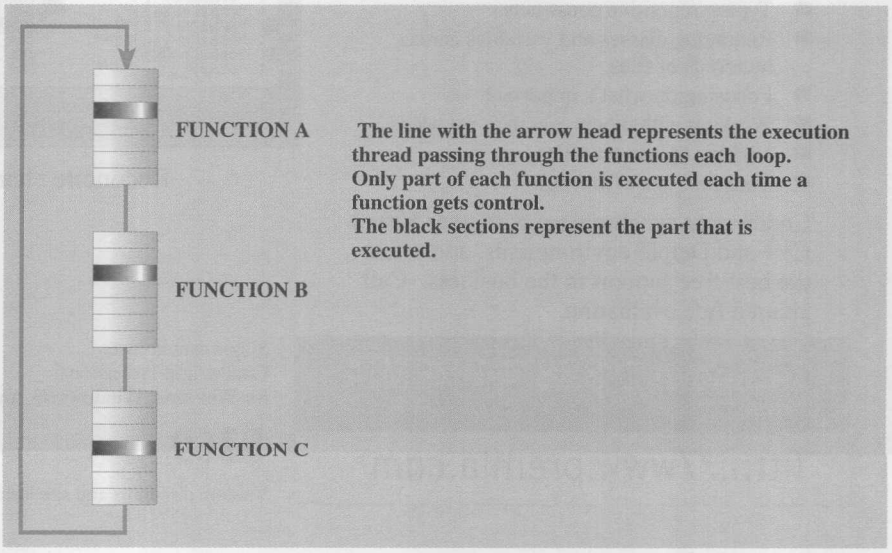
The use of state machines for each function or task enables the software to be small yet respond quickly to real world events. Tasks include key debouncing, communications, LED flashing, and so on. Each function consists of a number of steps or states. Each individual state is a reaction or action that requires only a few instructions to perform. The entire program consists of passing control to each function in turn. Each function only executes part of itself to act on or react to one step. The result is that the system seems to be doing everything at the same time. Consider Figure 1, illustrating the program looping through each function. Each function executes a small part of itself and passes control to the next function.

This article introduces a variety of ways this concept can be implemented, accompanied with snippets of code. The code contains comments to explain the instructions used. Please note that the code is neither totally accurate nor tested. It's purpose is to serve as illustration.

To clarify this concept and expand upon its use, let's consider an example of a large tank filling with water and emptying repeatedly. Our little computer will control this process.

The drawing in Figure 2 represents a large container with an input pipe in the upper left and an output pipe in the lower right. The input pipe is controlled by a valve as well as is the out-

**FIGURE 1**
*Program loops through each function.*



**The use of state machines for each function or task enables the software to be small yet respond quickly to real world events.**

put pipe. The valves are output devices in that the computer can open or close them. If the computer writes a 0 to a valve, it is opened and allows water to flow. If the computer writes 1 to the valve, the valve is closed and no water flows.

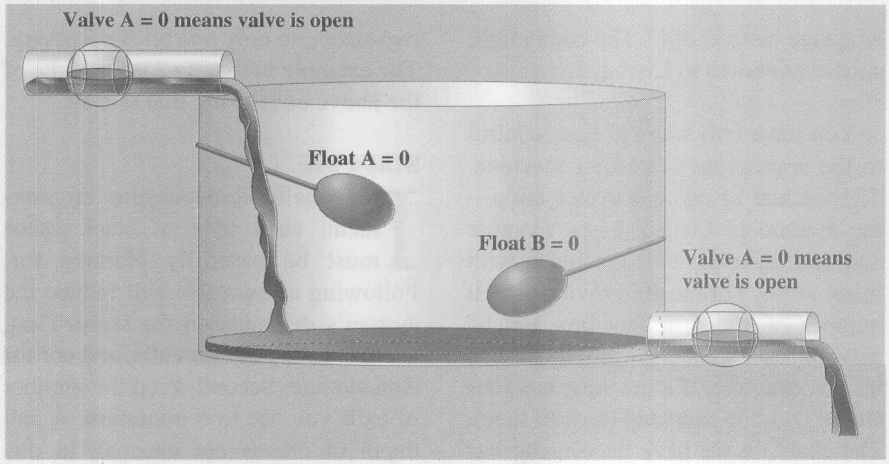
You can also see floats in the tank. Float A indicates the tank is full and Float B indicates the tank is empty. These floats are input devices that the computer is able to sense. If a float is up, it sends a value of 1 to the computer. If the float is down, it sends a value of 0 to the computer.

Our purpose in this task is to control the flow of water through the tank. We want to fill up the tank. Once full, we want to empty the tank completely. When it's empty, we want to fill it up again, and so on. The following C program describes how this task can be achieved:

```
void Main(void)
{
MainLoop;        //The program loops between here
  MachineOne( );
goto MainLoop;  //and here, calling MachineOne
                //each time
}

void MachineOne()
{
   //The switch statement passes control to one
   //of the states depending on the value in
   //StateVariable
   switch(StateVariable)
    {
case FILLING: //Here the StateVariable is equal
              //to FILLING
   if(FloatA = 1)//if float a = 1 (up) do this code
   {
        //The upper float moved up, start dumping
        ValveA=1;
        ValveB=0;
        StateVariable=DUMPING;
    }
break;
case DUMPING: //Here the StateVariable is equal
              //to DUMPING
      if(FloatB == 0)//If float B is 0 (down)
                     //do this code
```

**FIGURE 2**
*Tank of water as illustration of the repeated looping through of a program.*




```
    {
      //The lower float moved down, start
      //filling.
      ValveA=0;
      ValveB=1;
      StateVariable=FILLING;
    }
  break;
  }
}
```

Note these important features of this implementation:

- There is a main control loop which constantly calls the "machines"
- Most of the time nothing is going on, so a "machine" is merely executing the same "state" over and over again. The **if** statement in each "state" is executed and finds nothing has happened. This requires but a few instructions
- When an event does occur, only a few instructions are executed to take some action and possibly switch to a new state

The heart of the concept can be described like this: the main control loop calls routines to handle individual tasks in the system. Each routine consists of a state machine. Each state in the routine does one of two things. First, it checks to see if there is something to do. If not, control merely falls out of the function. If there is something to do, the state does it and changes the state variable to execute another state on the next pass through the main control loop. The process of checking to see if there is something to do requires two or three instructions. The something-to-do can require less than 15 instructions or about 15$\mu$s. Therefore, as many as 60 machines could be executed within 1ms.

## IMPLEMENTING STATE MACHINES

Let's continue our study of the state machine concept by using code suitable for microcontrollers. The 8051 family assembly language will be used. Our first task is to express the state machine in assembly language instead of C. The code might appear as shown in Listing 1.

**LISTING 1**
*Water tank program in assembly language.*

```
; Note >>> lines beginning with ';' are comments, not code
MainLoop:
 lcall Machine        ;execute MachineFunction
 sjmp  MainLoop       ;jump to MainLoop


Machine:
 ;These first few instructions transfer control to
 ;one of the states of this machine.
 mov DPTR,#JmpTbl     ;get address of the jump table
 mov a,StateVariable ;get value of StateVariable
 jmp @a + DPTR        ;jump to JmpTbl + StateVariable
JmTbl:
 ajmp FillingState    ;jump table
 ajmp DumpingState    ;jump table

FillingState :
 ;In this state float A is down and float B is up
 jnb FloatABit,DoNothingA     ;FloatABit not set,jmp to DoNothingA
 ;float A became 1, tank is full, change state to empty tank
 setb ValveABit       ;ValveA=1;
 clr ValveBBit        ;ValveB=0;
 mov a,#2             ;change state to dumping
 mov StateVariable,a ;change state to dumping
DoNothingA:
 ret

DumpingState :
 ;In this state float A is up and float B is down
 jb FloatBBit,DoNothingB  ;tank not MT, do nothing
 ;FloatBBit became 0, tank MT, change state to fill tank
 clr ValveABit        ;ValveA=0;
 setb ValveBBit       ;ValveB=1;
 mov A,#0             ;change state to filling
 mov StateVariable,A ;change state to filling
DoNothingB:
 ret
```

You have two ways to pass control to the appropriate state in a machine. The method of using a vector table—the method in Listing 1—is versatile and can be used with machines with many states. For machines with a small number of states, a few bits can be used to track what state the machine is in. For example, if a machine has four states, two bits can keep track of them. The example we have been using has two states, so only one bit is necessary. The example in Listing 2 is a rewrite of the above code using that one bit.

## STACK USE

In a small microcontroller environment, each byte of stack space must be carefully planned for. Following a few rules will reduce the system's demands on the stack. First, use the stack only for calls, and not for data storage. Second, keep the number of calls you use to a minimum. A call depth of one is not unusual. In this

**LISTING 2**
*Using one bit as a state variable.*

```
MachineFunction:
 jb StateVariableBit,DumpingState
FillingState:
 jnb FloatABit,DoNothingA    ;FloatABit not set,jmp to DoNothingA
 ;float A became 1, tank is full, change state to empty tank
 setb ValveABit              ;ValveA=1;
 clr ValveBBit               ;ValveB=0;
 setb StateVariableBit       ;change state to dumping
DoNothingA:
 ret ;return to caller of MachineFunction
DumpingState:
 ;In this state float A is up and float B is down
 jb FloatBBit,DoNothingB  ;tank not MT, do nothing
 ;FloatBBit became 0, tank MT, change state to fill tank
 clr ValveABit            ;ValveA=0;
 setb ValveBBit           ;ValveB=1;
 clr StateVariableBit     ;change state to filling
DoNothingB:
 ret ;return to caller of MachineFunction
```

**LISTING 3**
*Flashing an LED.*

```
FlashLed:      ;The main loop calls this label
   jb LedStateBit,LedOff ;if LedStateBit = 1, jmp LedOn
   ;if LedStateBit = continue here
   mov A,FlashTimer    ;get timer to see if its zero
   jnz KeepLedOff      ;not zero, do nothing
   mov TimerA,#10      ;is zero, restart timer
   setb   LedStateBit  ;and change state
KeepLedOff:
   ret                 ;return to caller
LedOn:
   mov A,TimerA        ;get timer to see if its zero
   jnz KeepLedOn       ;not zero, do nothing
   mov TimerA,,#10     ;is zero, restart timer
   clr   LedStateBit   ;and change state
KeepLedOn:
   ret                 ;return to caller
```

method of programming, subroutines are not often used. Special needs always exist, so a stack with a call depth of three is acceptable. With only 64 bytes of RAM available, keeping the stack small is highly desirable.

Normally the stack is also used for handling interrupts. That is, when an interrupt occurs, the interrupt routine saves all important registers, which can amount to more than 10 bytes. The space need not be reserved for interrupts if you use a technique that disables interrupts during normal processing. Then interrupts are allowed during those times when nothing is happening and the registers are essentially not used. Consider this piece of code:

```
MainControlLoop:
   lcall Machine1
   setb E0   ;allow interrupts
   clr  E0   ;enable interrupts
   lcall Machine2
   setb E0   ;allow interrupts
   clr  E0   ;enable interrupts
   lcall Machine3
   setb E0   ;allow interrupts
   clr  E0   ;enable interrupts
   sjmp  MainControlLoop
```

Interrupts are handled between calls to machines. Because the registers are not being used, they don't need to be saved. This technique works because each machine only requires about 15$\mu$s to get its job done. The interrupt latency, then, is 15$\mu$s, which is acceptable in most systems.

## BYTE TIMERS

Timing events within your program is one of your more complex tasks. Often timers are required for a variety of uses, from communication time-outs to debouncing keys and flashing LEDs. The time control method presented here is simple. In this method, a byte is used for each software timer we need. When a timer interrupt occurs, each byte is examined to see if it is 0. If it is not 0, it is decremented:

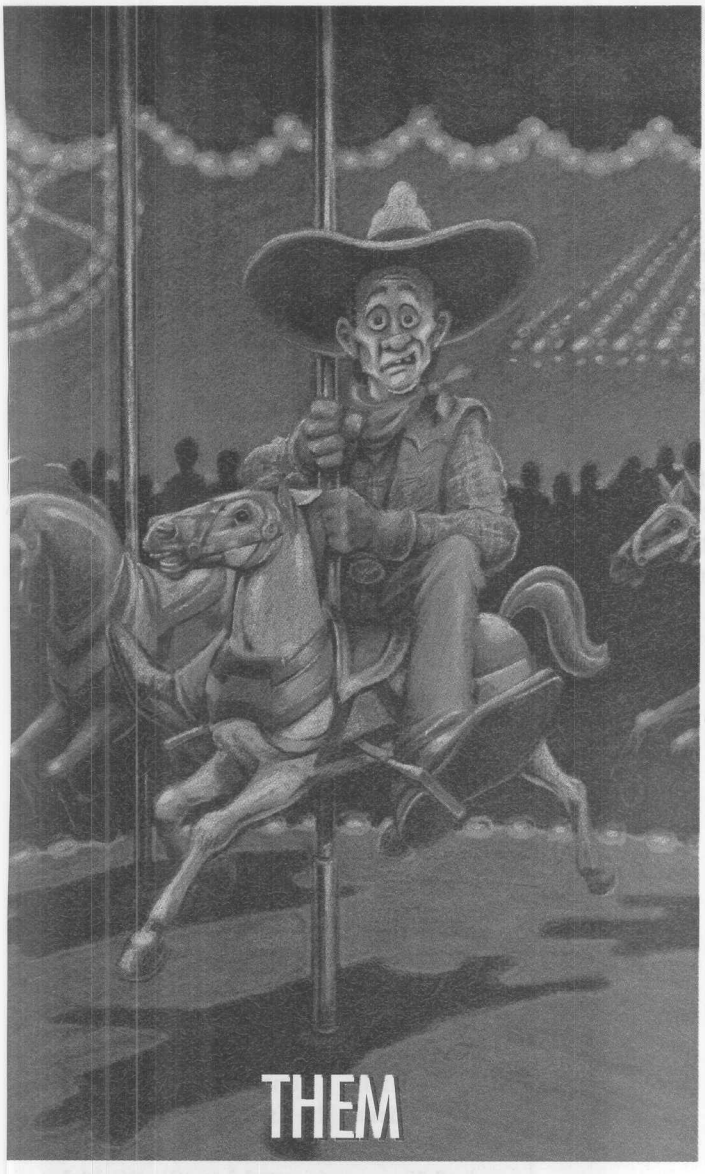
```
TimerInterrupt: ;Execute this when interrupt occurs
  mov a,TimerA  ;Get timer byte into register a
  jz ItsZero    ;Jump if TimerA is 0
  dec TimerA    ;If it's not 0, subtract 1 from it
ItsZero:
  iret
```

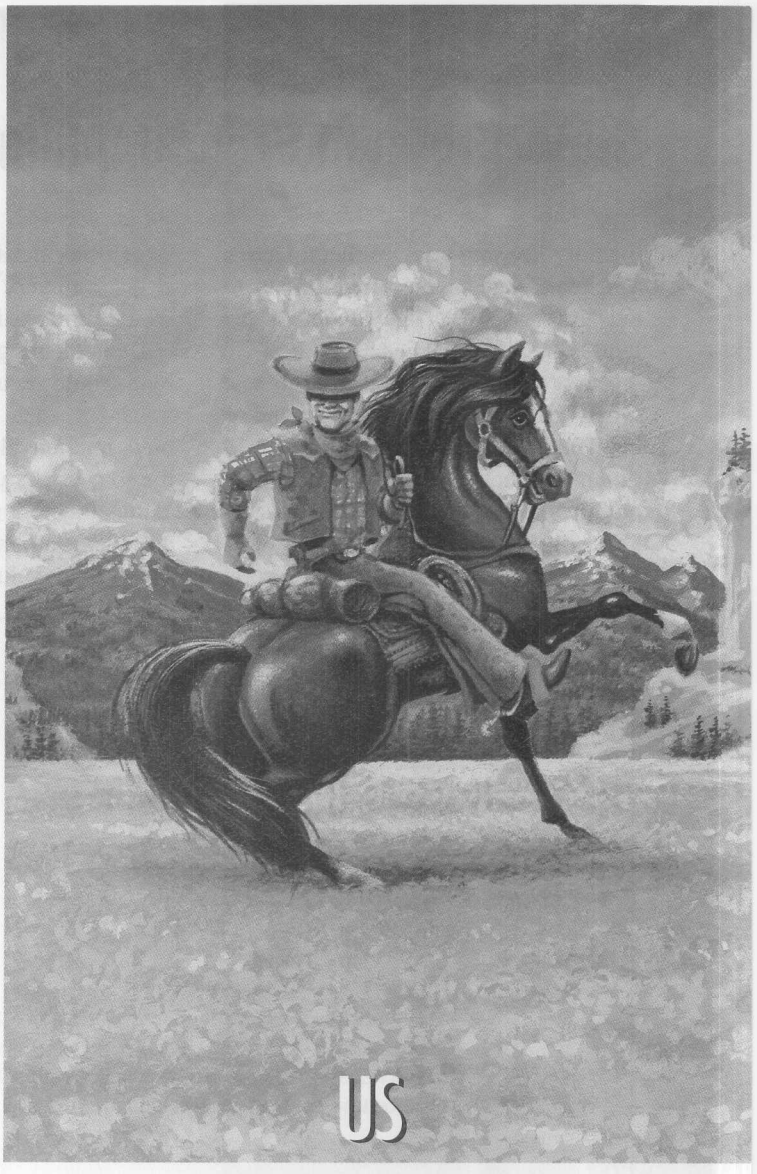
If the timer interrupt occurs every 100ms and a timer is set to, say, five, then the timer will time out or become zero between 400 and 500ms.

When a task wishes to use a timer, it stores a value in the timer value. Then that task continues to test that value for zero. When it goes to zero, the task continues, knowing an appropriate time has passed. To demonstrate this, let's make a machine that flashes an LED (see Listing 3).

## MULTIPLE TIMERS IN ONE BYTE

With the timer scheme just presented, each software timer requires one byte. If you have a number of items to time, you could expend a lot of precious




# CHOOSE YOUR HORSES WISELY.

*Real-Time Software for Embedded Applications*

Looking to ride into the future with an embedded software vendor? You have two choices: follow the same old well-worn path, or saddle up with an innovator... Accelerated Technology, Inc.

Instead of making you go in circles, we continue to blaze new trails in the embedded software frontier, with innovative approaches like offering our real-time kernel, networking, file, and debug products for *any* CPU and *with* source code. Plus, we offer them *without any run-time royalties.*

This pioneering spirit is moving us into previously uncharted territory. We're finding our own space, instead of following the herd. And we'll continue to explore new horizons with exceptional products offered at surprisingly reasonable prices, along with customer support that's in a realm all its own.

Break free of the same old run-around. Call Accelerated Technology, Inc. today at 1-800-468-6853 for more information on our company and our products. *We'll take you as far as you want to go.*

***Processor Support:***
- *ARM6/7 • ARM Thumb • StrongARM*
- *SHx • H8/300H • V25 • PowerPC*
- *Am29K • i960 • IDT 30xx • IDT 46xx*
- *LR330x0 • VR4x00/5000 • V830/850*
- *PR30100 • SPARClite • NEC78K • 680xx*
- *683xx • 68HC11/16 • ColdFire • MN10200*
- *80x86RM • 80386/486PM • DOS • M3770x*
- *TMS320C3x/40/2x/5x • 5600x DSP*

Accelerated Technology
INCORPORATED

*source code, no royalties, any CPU... it just makes sense*

720 Oak Circle Dr. East Mobile, Alabama 36609 334.661.5770 Fax 334.661.5788 800.468.6853 E-mail: sales@atinucleus.com http://www.atinucleus.com

**CIRCLE # 37 ON READER SERVICE CARD**

memory keeping track of them. Here's a scheme that enables four timers to exist in one byte. The layout of the byte is AABBCCDD.

AA represents two bits for one timer, BB, two bits for the next, and so on. The idea is that each of these 2-bit timers is checked to see if it is zero. If not, the timer is decremented by one. If a timer is set to three or binary 11, then the timer will time out or go to zero in three timer ticks. If the interrupt timer is ticking every 50ms, then this timer would time out in 100 to 150ms.

Decrementing each 2-bit field in the byte would require about five instructions for each timer. However, one bold method checks and decrements the four timers using four instructions. In this method, a timer byte is used as an index into a table to retrieve a new timer byte value. The values in the table are such that the appropriate non-zero 2-bit fields are decremented while those that are zero remain zero.

The major problem with this is that 256 bytes of ROM are required. The real gain is that three bytes of RAM can contain 12 timers. With the byte method, 12 bytes would be required. Often you will have plenty of ROM but little RAM, so this scheme can be valuable. The 2-bit timers are set with an `OR` instruction and tested using an `AND` instruction.



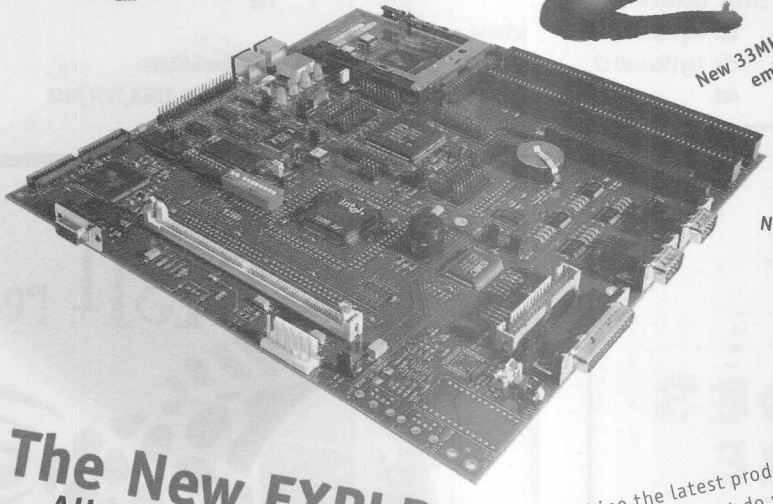
## USING A LARGE NUMBER OF TIMERS

Decrementing all timers with each occurrence of an interrupt would be desirable, for the process could take longer than 15$\mu$s if there were a large number of them. Normally the timers need not be decremented on the same pass. In this scheme, the timers are not decremented when the timer interrupt occurs. Instead, the timer interrupt sets `TimersState` equal to zero, then when the main control loop calls `HandleTimers` the timers are managed.

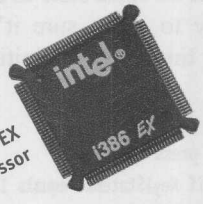
```
HandleTimers:  ;main control loop calls this label
  mov DPTR,#TimerCode  ;get address of TimerCode
into DPTR
  mov a,TimersState    ;get offset into TimerCode
  jmp @a+DPTR      ;Jump to the appropriate state
TimerCode;
  mov TimersState,#INDEX_TO_TIMERA ;set up for next
state
  ;decrement timer here
  ret                  ;done with this timer
  mov TimersState,#INDEX_TO_TIMERB
  ;decrement timer here
  ret
  ret                  ;this is idle state
```

In this example, when the `INDEX_TO_TIMER` value is added to the `TimerCode` address, the jump indirect (`jmp @a+DPTR`) instruction will pass control to the appropriate routine. That routine handles the timer and selects the next `INDEX_TO_TIMER` value. When all timers are handled, only the last state is executed, which simply returns until the timer interrupt resets the state variable to zero.

# radical technology embedded



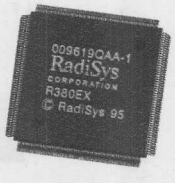
New 33MHz Intel386™ EX embedded processor




QNX RTOS with Photon microGUI®

QNX® The Leading Realtime OS for PCs

New RadiSys R380EX system controller




## The New EXPLR2:
### All the hottest embedded technologies on one handy eval board!

Combine the latest products from the leaders in embedded PC technology and what do you get? Faster time-to-market. More functionality with less hardware. A longer production life for your product. And a lot less work.

Intel's EXPLR2 makes your design work easy. All the hardware you need is fully integrated, so you can start prototyping right away. The EXPLR2 also comes with QNX, the realtime OS that delivers high-end performance, even on low-cost hardware. Exceptionally small and fully scalable, QNX gives you a POSIX-certified environment *and* a full-featured GUI in under 1MB of memory!

The EXPLR2 comes with something else. Commitment. With their long-term commitment to embedded applications, Intel, RadiSys, and QNX Software Systems are the right companies to give your designs a real future. **Radical!**

**For your free reference design kit, surf: www.explr2.com or call: 800 862-1883**
**(to order a board, contact your Intel distributor)**

New


**One-Stop Internet Toolkit!**

**Tools**, demo apps, source code, hardware... **everything** you need to start **today!**
**Intel product code: EX386INTRNETUS**

**CIRCLE # 38 ON READER SERVICE CARD**

# State-Oriented Programming

## SWITCH DEBOUNCING

Let's take a look at a real task and apply this technique to debouncing keys and switches. The debounce strategy is to accept a key closure immediately as a closure but wait for a while to allow bounce to occur before checking to see if the key has been opened. The same occurs when the switch opens. We treat the initial opening as such, but allow some time to pass before checking to see if the key might be closed again. Those engineers who are worried about false signals due to noise on the wire from the key can use an extra state to double check the key to make sure it's still closed a short time after the initial key closure.

```
;Debouncing a Key press
;Key 1 is closed if Key1StateBit equals 1
;Key 1 is open if Key1StateBit equals 0
;Key1Input is connected directly to key or
switch
KEY1_TIMER_SET_MASK equ 0c0h
;set mask for 2-bit timer
KEY1_TIMER_TEST_MASK equ 3fh
;test mask for 2-bit timer
DoKey1: ;Main control loop calls DoKey1
   ;jump to appropriate state
   jnb  Key1StateBit1,Key1ClosedStates
   jb  Key1StateBit2,Key1OpenState
   sjmp Key1JustOpenedState
Key1ClosedStates:
   jnb  Key1StateBit2,Key1ClosedState
   sjmp Key1JustClosedState

Key1OpenState:
   ;if Key1Input =1, jmp to KeyJustClosed
   jb Key1Input,KeyJustClosed
   ret   ;key not closed, do nothing
KeyJustClosed:
   orl KeyTimer,#KEY1_TIMER_SET_MASK
;start debounce timer
   set Key1StateBit1        ;closed state
   clr Key1StateBit2        ;just closed state
   ret

Key1JustClosedState:
   mov a,#KEY1_TIMER_TEST_MASK
;check if timer timed out
   anl a,KeyTimer
   jz CloseDebounced   ;if it did do jump
   ret              ;if not, just return
CloseDebounced:
   set Key1StateBit2
;done debouncing, go to closed state
   ret

Key1ClosedState:
   ;if Key1Input =0, jmp to KeyJustOpened
   jnb Key1Input,KeyJustOpened
   ret   ;key not closed, do nothing
KeyJustOpened:
   orl KeyTimer,#KEY1_TIMER_SET_MASK
;start debounce timer
   clr Key1StateBit1        ;open state
   clr Key1StateBit2        ;just opened state
   ret

Key1JustOpenedState:
   mov a,#KEY1_TIMER_TEST_MASK
```

# LynxOS (actual size)

**Only LynxOS is fully scalable across all of your communications products.**

# LynxOS (actual size)

Now you can standardize on a single operating system for your smallest to your largest embedded, real-time applications. From a full featured Unix-compatible, POSIX-compliant system down to a small 28 Kbyte microkernel – and everything in between, flexible LynxOS™ can precisely meet your embedded, real-time performance and deterministic requirements.

LynxOS employs Kernel Plug-Ins™ that can be added to your system when additional functionality is required. You save time and money through software re-use across product lines and portability across all popular hardware platforms.

Our PosixWorks® development environment streamlines software development to insure your product is delivered on schedule. And because you can use one operating system across your entire range of products, this single set of development tools provides maximum productivity for your engineering teams.

**LEARN MORE ABOUT LYNXOS IN OUR FREE REAL-TIME NEWSLETTER.**


(Not actual size)

**Call 1-800-255-LYNX**

**www.lynx.com**

***Committed to your success in real-time.***



Lynx REAL-TIME SYSTEMS, INC.

**CIRCLE # 40 ON READER SERVICE CARD**

```
;check if timer timed out
   anl a,KeyTimer
   jz OpenDebounced   ;if it did do jump
   ret                ;if not, just return
OpenDebounced:
   set Key1StateBit2
;done debouncing, go to open state
   ret
```

In this routine, you can see the timer in use, which of course is decremented elsewhere in the program, and you can see that each step of the machine requires very few instructions to execute during each pass through the main loop.

## DISPLAY DEVICES

Our next example illustrates writing data to an array of LEDs (see Figure 3). Below is a simplified model made up for the purposes of this explanation. In this model, bits are fed into the DAT line one at a time. As the bits enter, they all move around like a bucket brigade. The dots inside the dotted box represent LEDs that make up a 5-by-7 character array. If a one bit falls on a dot, that LED is lit. If a zero falls on a dot, it is not lit. The CE (chip enable), CLK (clock), and DAT (data) lines are manipulated to get the chips into the array. The CE line is brought high to start sending bits into the array. To move a bit into the array, the CLK line is raised to one. Then the DAT line is raised to one if a bit is to be moved in, otherwise the DAT line remains zero if a zero bit is to be moved in. Then the CLK line drops to zero to tell the array a bit is on the DAT line. That process is repeated for each bit. Then the CE line is dropped to zero, which tells the array all bits are in and the one bits can be turned on.

The following code is designed to clock one bit in every pass through the main control loop. Note that to start the machine working the code invoking the routine puts the bits to make a character into Data0-4 and sets `LedState`, the state variable, to `LED_START_STATE`. When the routine is done entering the bits, the state variable equals `LED_IDLE_STATE`.

```
;NOTE: code to vector to the state labels is not
;displayed here
LedIdleState:
  ;do nothing until someone changes state vari-
able to   ;LedStartState
  ret

LedStartState:
;Begin to display character in Data0-4
  mov r0,#Data0 ;get index to first byte
  setb CE    ;raise chip enable to write bits to
LED array
  mov LedState,#LED_BIT_STATE
;switch to display bit state
  ret

LedBitState:
  setb CLK
;raise the clock letting LED know a bit is coming
  mov a,@r0
;retrieve bits to write to LED array
  rlc a         ;rotate a bit into carry flag
  mov DATA,c
;write the carry flag to the data out pin
  mov @r0,a
;save the bits to write to LED array
  clr CLK
;drop clock informing LED the bit is done
  clr DATA
;clear the data out pin to ready for next pass
  djnz BitCnt,StillBitsToDo
;decrement bits to write
                ;& jump if not done
  mov LedState,#LED_BYTE_STATE
;done writing, change state
StillBitsToDo:
  ret

LedByteState:
;here we check to see if there are more bytes to do
  inc r0
;compare index to end of data area, jump if not equal
  cjne r0,#DATA4+1,StillBytesToDo
  mov a,#LED_DONE_STATE ;we are done so
  mov LedState,a      ;change state
  ret
StillBytesToDo:
  mov BitCnt,#8
;more to do so reset bit count, do it again
  mov LedState,#LED_BIT_STATE
;done, change state to send bits
  ret

LedDoneState:
  clr CE
;drop chip enable causing character to be dis-
played
  mov LedState,#LED_IDLE
;go to idle state because we are done
  ret
```

This routine would display a character in about 10ms. This assumes that for most functions the main control loop calls have nothing to do when they are called.

## COMMUNICATIONS

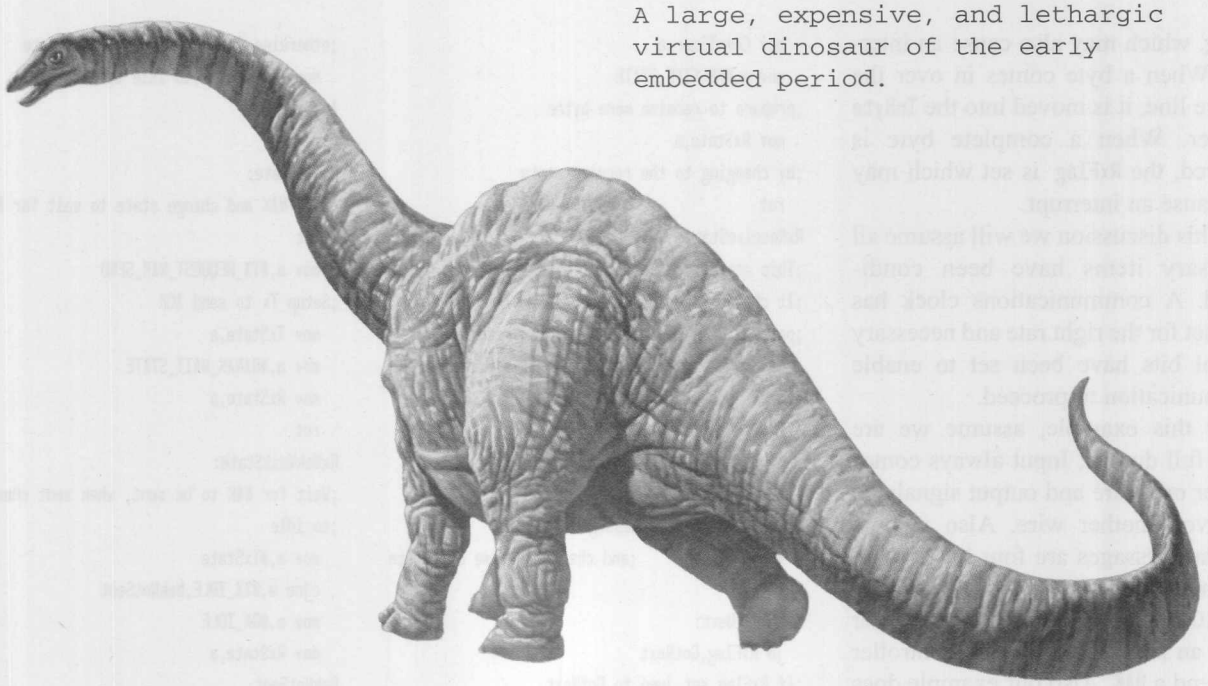
Many microcontrollers support some sort of communication hardware. This could range from IIC, SPI, SCI, or other serial communication systems. This hardware handles the details of communication while the software acts in a supervisory role. The hardware provides registers for the software through which control is effected.

Here I describe a rough model of the communications environment that contains common elements of these systems. Then I'll present a sample of code that handles communication with the concepts presented in this article. Here's how the registers and hardware flags may be organized:

- `InByte`: A register in memory which receives a byte from the outside world
- `OutByte`: A register in memory which sends a byte to the outside world
- `RxFlag`: A bit when set by the hardware indicates `InByte` has received a byte
- `TxFlag`: A bit when set by the hardware indicates `OutByte` has been sent

To send a byte out of the system, the software would put a byte into `OutByte` and the hardware, recognizing the new data, begins to send the data serially out over the transmission line. When the byte is sent, the processor sets the

**RTOS•sau•rus** (ar-tos-sor-es) n.
A large, expensive, and lethargic virtual dinosaur of the early embedded period.



# Don't let extinct kernels take your product development into the Ice Age...

Express Logic can lead you to a new age of embedded systems development with ThreadX™—a real-time kernel designed for real-time applications like yours.

*Size*. ThreadX has a microscopic footprint—typically requiring under 3Kbytes of instruction-area memory.

*Speed*. Because of its advanced picokernel™ architecture, ThreadX achieves the fastest possible performance.

*Intelligence*. ThreadX is top of the order. It features preemption-threshold™—a completely new technique to reduce context switching. For more information, ask us for a free whitepaper!

*Easy to use*. ThreadX is a snap to use! Check out our API to see for yourself. In addition, you get full C source code should you need to customize it.

*Processor support*. You name it. ThreadX supports most popular processors, and we're adding new ones all the time.

*Business friendly*. Absolutely. ThreadX kernels are offered at reasonable prices and—most importantly—without any run-time royalties!

To find out how you can speed up your product development with ThreadX, call Express Logic today.

EXPRESS LOGIC

toll free **1-888-THREADX**
1-619-674-6684
info@expresslogic.com
www.expresslogic.com



**CIRCLE # 41 ON READER SERVICE CARD**

TxFlag, which may also cause an interrupt. When a byte comes in over the receive line, it is moved into the InByte register. When a complete byte is received, the RxFlag is set which may also cause an interrupt.

In this discussion we will assume all necessary items have been conditioned. A communications clock has been set for the right rate and necessary control bits have been set to enable communication to proceed.

For this example, assume we are using full duplex. Input always comes in over one wire and output signals go out over another wire. Also assume that the messages are four bytes long. This message must be received in less than 100ms. If that is so, the controller sends an ACK; otherwise, the controller will send a NAK. Also our example does not use interrupts but constantly examines RxFlag and TxFlag, taking action when they are set. Only the input or read routine will be presented.

The philosophy is the same as before; a function, named GetMessage, handles the job and is called from the main control loop. This function is a state machine. RxState is the state variable. The routine GetMessage accepts 4 input bytes placing them into Data0, Data1, Data2, and Data3, which are locations in RAM.

```
GetMessage:
;NOTE: the code to vector to the state labels
;has been left out for brevity
RxIdleState:
;this state is constanly executed while waiting
;for input
;When it detects an input byte it receives the
;byte and prepares to receive the other bytes.
  jb RxFlag,GotInput
;If RxFlag set jump to GotInput
  ret ;just return to caller, no input yet
GotInput :
  mov a,RxByte
;get input byte into register a
  mov r0,#DATA0
;initialize index to message area
  mov @r0,a
;move register a into message area
  inc r0        ;point at next MT slot
  mov a,#COMM_TIMEOUT  ;fire up time out timer
  mov CommTimer,a
  mov a,#RECEIVE_STATE
;prepare to receive more bytes
  mov RxState,a
;by changing to the receive state
  ret
RxReceiveState:
;This state receives input bytes
;It checks the timer to see if to much time has
;passed. When all bytes have been read it
;changes state to send an ACK mov a,CommTimer
;check to see if we timed out
  jnz CheckForNext
;jump if we have not timed out
  mov a,#RXTIMEOUT
;fall to here with message timeout
  mov RxState,a    ;and change to time out state
  ret
CheckForNext:
  jb RxFlag,GotNext
;if RxFlag set jump to GotNext
  ret          ;if not set just return to caller
GotNext:
;Here we received another byte
  mov a,RxByte    ;move byte to register a
  mov @r0,a
;move register a into message area
  inc r0
;point at next byte in message area
  cjne r0,#Data3+1,Next
;compare index to input data (r0)
;and jump if not equal to Data3+1 address
  mov a,#RXACK_STATE ;done receiving bytes,
  mov RxState,a     ;change state to send ACK
Next:
  ret
RxAckState:
;Send ACK and change state to wait for ACK to be
sent
  mov a,#TX_REQUEST_ACK_SEND
;Setup Tx to send ACK
  mov TxState,a
  mov a,#RXACK_WAIT_STATE
;change read state to
  mov RxState,a            ;wait for ACK sent
  ret


RxAckWaitState:
;Wait for ACK to be sent then change state to
;idle state
  mov a,#TxState
;if Tx send state NE to idle
  cjne a,#TX_IDLE,AckNotSent ;ACK is not sent
  mov a,#RX_IDLE
;otherwise fall here and change state
  mov RxState,a ;to idle state
AckNotSent:
  ret
RxNakState:
;Send NAK and change state to wait for NAK to be
;sent
  mov a,#TX_REQUEST_NAK_SEND
;Setup Tx to send ACK
  mov TxState,a
  mov a,#RXNAK_WAIT_STATE
  mov RxState,a
  ret
RxNakWaitState:
;Wait for NAK to be sent, when sent change state
;to idle
  mov a,#TxState
  cjne a,#TX_IDLE,NakNotSent
  mov a,#RX_IDLE
  mov RxState,a
NakNotSent
  ret
```

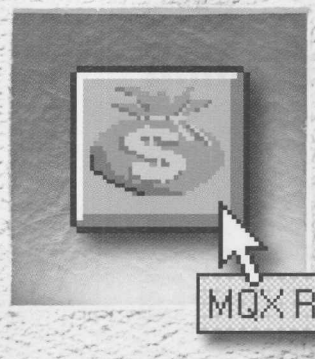
## SMALL FEATURE MICROCONTROLLERS

In general the microcontrollers I've discussed here have been small but have still had several important hardware extras. Here's how to deal with the situation in which you have few or no luxuries at all. Consider, for example, a microcontroller without stack hardware, indirect addressing capability, or communication specific hardware.

## LITTLE OR NO STACK

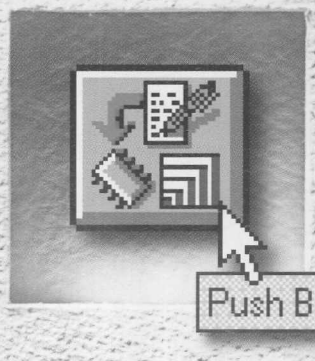
Some small microcontrollers have limited stack capability or none at all. The method I present in this article reduces the need for the stack. The main control loop used in all of the examples uses calls to all of the functions. The calls could be eliminated by simply having all the functions in line. Instead of returning from the function when it completes, the program could jump to the end of the function where the next function would begin. The stack is also used in many implementations as calls to common routines. The approach taken in this article does not use common routines since each state requires so few instructions. If a piece of code is used in more than one place it is simply duplicated. The reduced need for stack

# Point, Click, Deliver




- Delivers significant cost savings
- Single and distributed processor operation
- Scaleable microkernel architecture
- Extensive embedded I/O components such as TCP/IP, PPP, LAPB, HDLC, CAN...

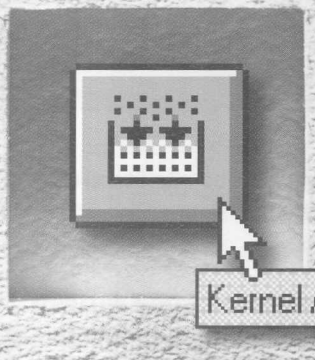
**68302 Kernel**




**68040 Kernel**

- Integration that works
- Easy-to-use point and click interface to all tools
- Project oriented development
- Automated build process

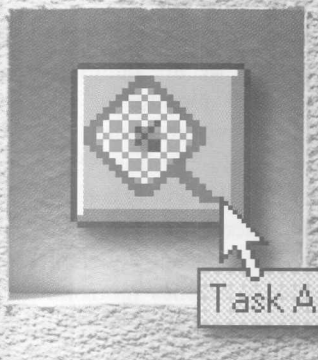
**68EC000 Kernel**




**CPU32 Kernel**

- Totally integrated compiler, assembler and linker/locator
- Target specific extensions and libraries support the whole 68xxx family
- Outstanding code quality
- Hardware and software floating point

**68340 Kernel**




**68360 Kernel**

- Write in C debug in C
- Full trace including C, assembly and stack
- Kernel aware debugging
- I/O simulation
- Available for ROM monitor, simulator, BDM and In-Circuit Emulator (ICE)

**68332 Kernel**

**68010 Kernel**


**68060 Kernel**

- Engineer tested Motorola approved
- Proven product line since 1980
- Toll free customer support
- Continued leading-edge development
- Watch for ColdFire® and Power PC®

The competition is fierce. You are under pressure to get your product out fast and at the lowest cost. What you need is the INTERTOOLS KERNEL SOLUTION. It's the **only fully integrated 68xxx toolset with a Precise/MQX® kernel** and a host of easy to use features found nowhere else.

**The Best 68xxx Kernel Solution**

When it comes to helping you get to market faster – with a better built, better designed product – nobody beats TASKING.

No wonder TASKING is the world's leading choice for software development tools across industry standard computing platforms; Windows® 95, Windows® NT, UNIX. Throughout the world, more embedded applications have been developed using TASKING tools than anyone else's.

To find out how to deliver your product to market faster call us at **1-800-458-8276**, or visit us at **http: www.tasking.com**

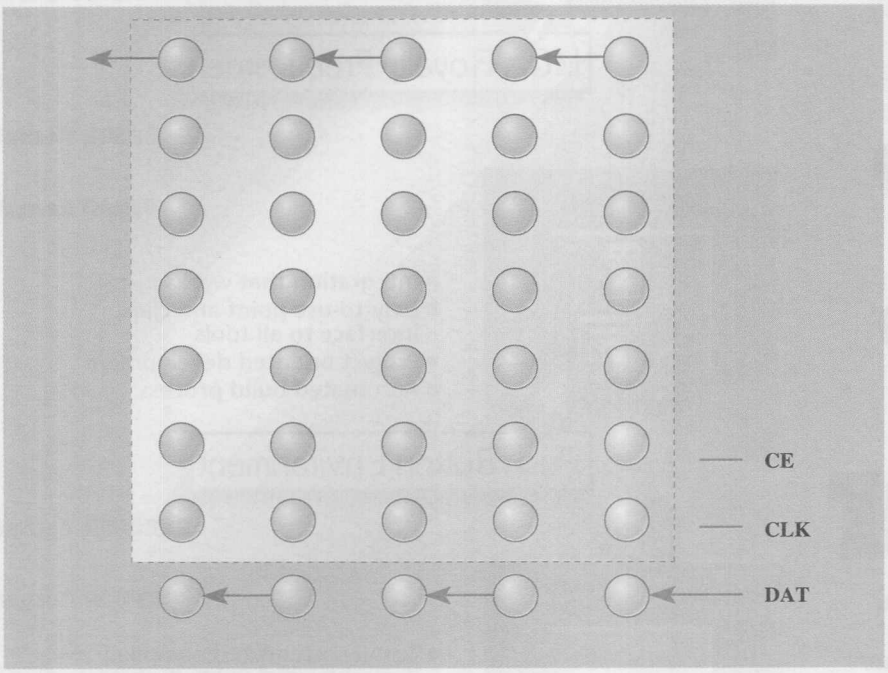
TASKING

Quality Development Tools Worldwide

*BSO/TASKING and Intermetrics Microsystems have joined to become TASKING*

**CIRCLE # 42 ON READER SERVICE CARD**

**FIGURE 3**
*An array of LEDs.*




space enables those microcontrollers with limited stack to use it for very important functions.

## NO INDIRECT ADDRESSING

The method presented here depends on the ability to use vector tables (via indirect addressing) to vector to the appropriate state. Here is an alternative. The state variable is decremented until it is zero. At that point the appropriate state is jumped to. Here is a code snippet showing this:

```
AFunction:

;main control loop calls this function
   mov a,StateVariable
;get state variable into register a
   jz State0
;if register a is 0 jump to state 0
   dec a
;subtract 1 from the register
   jz State1
;if reg a is now 0 jump to state 1
   dec a              ;and so on
   jz State2
```

The state getting control can then put a value into `StateVariable` to cause another state to be executed on next pass.

## COMMUNICATION WITHOUT COMMUNICATION HARDWARE

Now consider a microcontroller that does not have the hardware to support communications. Assume we are building a "machine" that will monitor an input line and accept asynchronous 8-bit messages commonly found in RS-232 communications. We don't need timer interrupts: the timer can be set up to run free. It can be constantly incremented so that when it reaches its maximum value it will wrap to zero. We'll call the routine to handle all of this `DoComm`. It's called between each function call in the main control loop. The main control loop might appear as follows:

```
MainControlLoop:
   call   KeyBoard
   call   DoComm
;Do input bit manipulation
   call   Heater
   call   DoComm
;Do input bit manipulation
   call   GetMessage
   call   DoComm
;Do input bit manipulation
   jmp MainControlLoop
```

Notice the `GetMessage` function, which is similar to the communication function we discussed earlier. It's another task, just as it was before. Before, however, the hardware was getting the data for it to handle. Now the `DoComm` function handles this task. Timing is the critical aspect of this task and is handled as follows. Each state only executes about 15 instructions which should require about 15$\mu$s for each function. Then every 15$\mu$s, or after each function, we call `DoComm`. Upon entry `DoComm` loops waiting for the exact time or "point of examination" necessary for its processing. The precise time is in `DesireTime`. The code might appear:

```
DoComm:
DesiredTime equ R4
;this code snippit assumes an 8048 type
architecture
DoCommA:
;loop to here to wait for right time
  mov a,T
;get time ticks from reg T into a
  ;8048 has no compare, use xor,complement
  ;for that
  xrl a,DesiredTime
;compare present time to desired time
  cpl a
;compare present time to desired time
  jnz DoCommA
;not yet, jump to DoCommA to try again
;fall here with time to handle communications
  add a,#COMMTIME  ;set up to get next comm
handle time
  mov DesiredTime,a
;save the next desired time
  ;continue processing communications
```

There are two critical times involved with the reading of an 8-bit byte. One time is the time to accept one bit. We can call that the *bit time*. The other time is how often we read the input line to check for that bit. That time is called the *point of examination* or *ticks* in this

# THE DEEP C APPROACH TO EMBEDDED DEVELOPMENT

***IAR Systems dives deeper in embedded systems development tools than any other supplier. With 15 years of proven record in providing high quality development tools for micro-controllers, IAR Systems is the only original manufacturer of such tools supporting the widest range of 8 and 16 bit embedded controllers.***

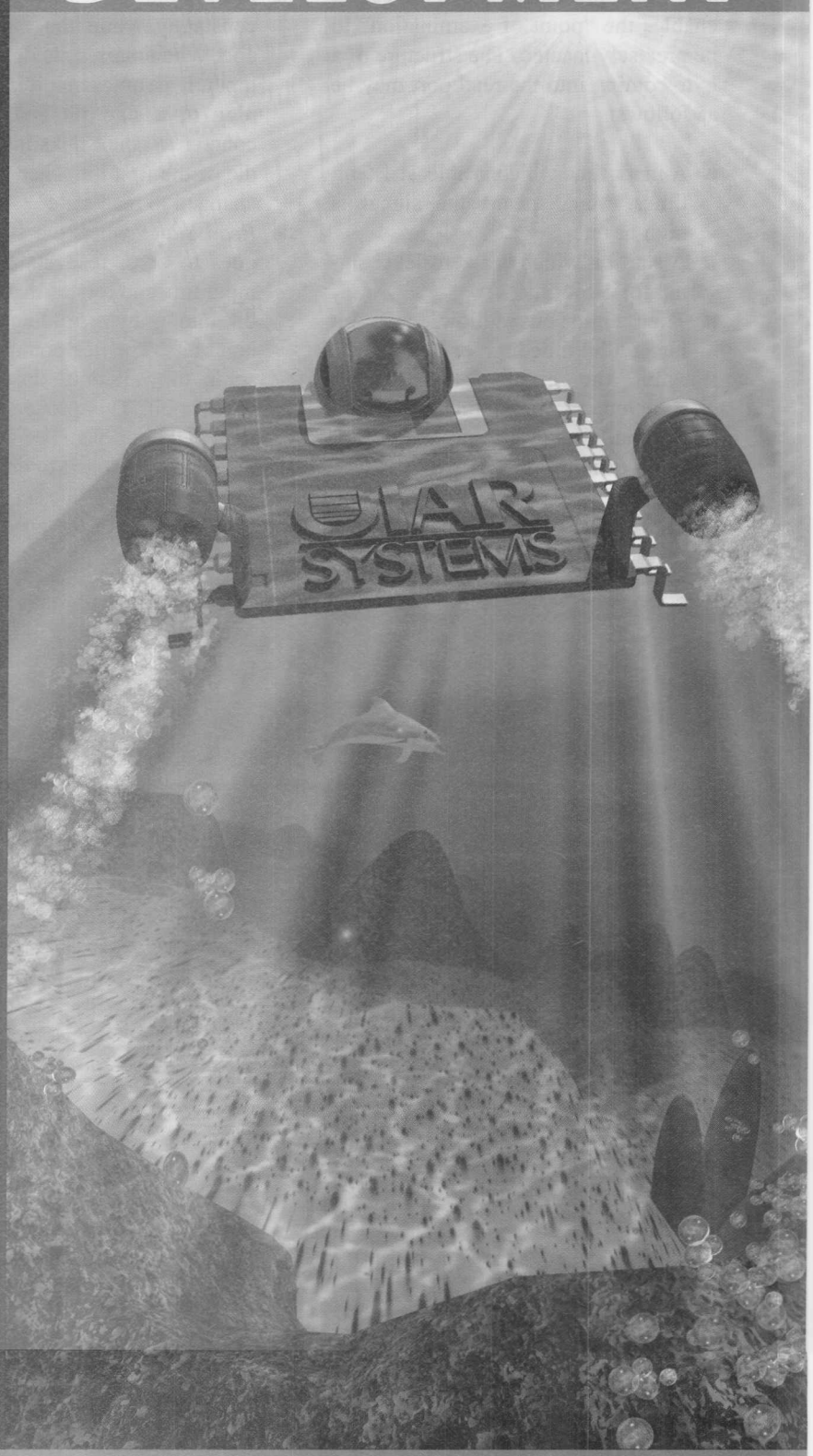
The renowned IAR C compiler offers 10 optimization levels, full ANSI compatibility including math & trigonometric libraries, floating point support, and reentrancy which makes it ideal to use with real time operating systems.

IAR's Embedded Workbench comes with an optimized C cross compiler, macro assembler, universal linker, librarian, C source level debugger, complete ANSI libraries and a fully integrated development environment under Windows™ (DOS is also supported).

**TARGETS SUPPORTED BY IAR C**

| INTEL | MOTOROLA | ZILOG |
|---|---|---|
| 8051* | 68HC11* | Z80* |
| 80196* | 68HC16 | Z180* |
| **HITACHI** | **MITSUBISHI** | **NEC** |
| 64180* | MELPS 7700 | 78000 |
| H8/300* (H) | MELPS 740 | 78200 |
| H8/500 | | 78300 |
| **TOSHIBA** | **PHILIPS** | **OKI** |
| TLCS 900 | 8051* | 65000 |

***Previously distributed under the *Archimedes* brand name.**




## CALL NOW FOR FREE DEMO OR MORE INFORMATION

## 1-800-427-8868

IAR SYSTEMS

IAR Systems Software, Inc.
One Maritime Plaza
San Francisco, CA 94111
Tel: 415-765-5500
Fax: 415-765-5503
Email: info@iar.com
www.iar.com

**CIRCLE # 43 ON READER SERVICE CARD**

# IAR SYSTEMS - NO ONE DIVES IN C DEEPER

document. The time from point of examination to point of examination is one-tenth of the bit time. Another way to put it is the point of examination occurs 10 times more often than a bit. The code presented above in DoComm enables the "point of examination" to be precisely located. The structure of a byte coming into the read port may be as follows:

- A mark time which is at least 1.5 bit times. Mark means the signal is high or one
- A start bit which is a zero-level signal for 1-bit time
- Then eight bits follow, each bit being a bit time high or low
- After the eight data bits are received, the input line may enter the mark state for another 1.5 bit times or more

DoComm must be written to accept this structure. As before, the DoComm function is divided into states each one doing its small share of the whole operation. The states could be as follows:

- *Mark state*: this state gets control constantly while the input line is one. When mark state detects a zero input, it assumes that it is the beginning of a start bit and sets up a counter to count ticks into the middle of the first bit. Then the state is changed to read state
- *Read state*: when the tick counter goes to zero, read state reads the input line and rotates that value as a bit into a byte holding the input eight bits. The tick counter is set up again to wait until the middle of another bit. This process continues until eight bits have been read. Then the state is changed to done state
- *Done state*: this state serves to notify other parts of the program that a byte has been read. The other function can read the state of this function, find the state variable equal to done state and take action to move the new byte in. One of the actions would be to set the state variable here to find mark state
- *Find mark state*: this state continues reading the input line until it is one for 1.5 bit times. Then it changes the state to mark state. The process is ready to read another byte

## DIVIDE AND CONQUER

The point I want to make in this article is that many tasks can be divided into smaller parts, each part of which is executed sequentially. This creates the illusion that all tasks are performed at the same time. This technique can be applied to almost any other task. Other functions that can be adapted to this technique include multiplying and dividing. These are loop-intensive tasks. Each loop could be a part of a math function that is executed each pass of the main control loop. Managing serial EEPROM is another application. Each pass through the main control loop could set up or finish a write. Often in manipulating these devices one must wait for some action to be completed within the part. With this technique, instead of waiting, the loop continues and other work is done.

Perhaps some day you will try this idea on a project that has limited capability and rewrite an old favorite routine, finding, like the author, that the performance of these little chips can be impressive. ESP

*Al Schneider has a BS in physics. He has been programming for 26 years, concentrating on embedded systems. Presently he is a contract programmer and has worked on many applications, including pacemakers, airplane guidance systems, industrial HVAC interfaces, and plastic and molten metal injection control systems. If you try using the technique described in this article, Al would like to hear from you. You can reach him at als@citilink.com.*

# THE LEADER IN EMBEDDED INTERNET PROTOCOLS

## TCP/IP

*100% RFC Compliant*

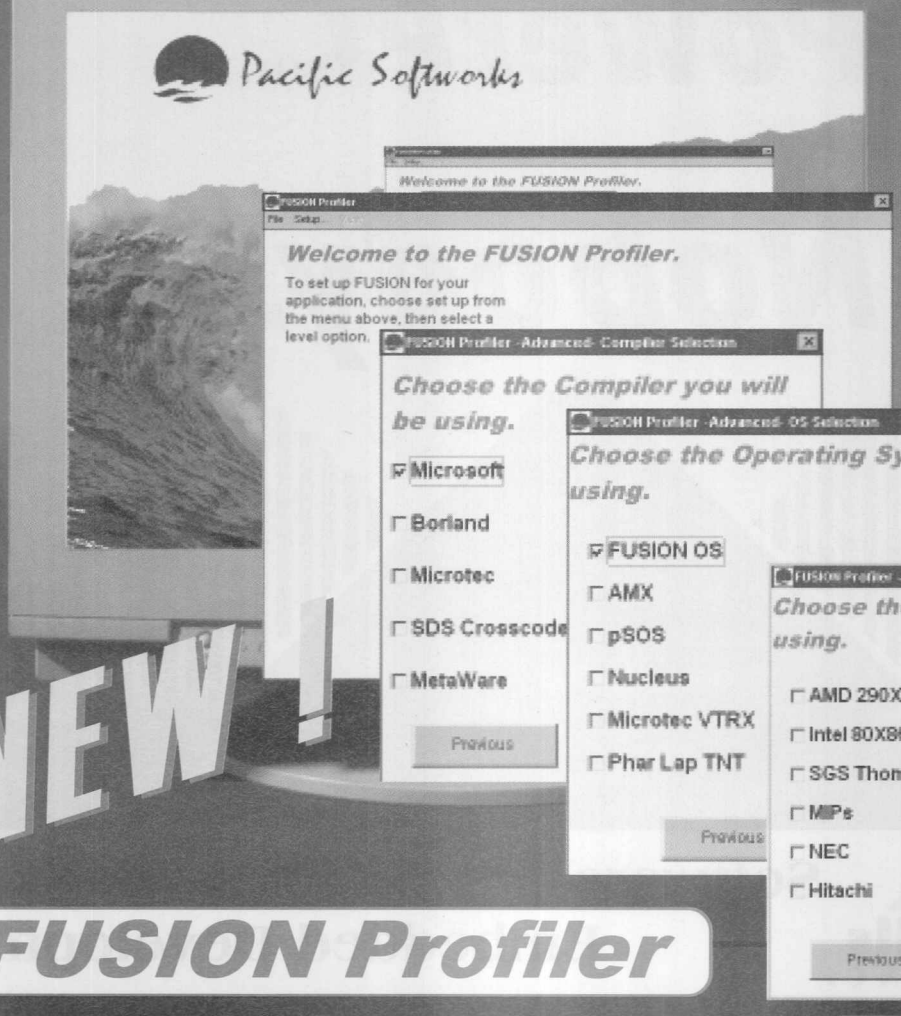
## Web Management

## SURF360/386

## FUSION

**SNMP**
**OSPF**
**RIP**
**PPP**

*Powerful internet Protocols Made Easy!*




**NEW!**

# FUSION Profiler

***FUSION, the most powerful Internet protocols speeds your time-to-market and saves you $$$$$ in development costs!***

***FUSION Profiler gives you point-and-click control over your TCP/IP development for Embedded systems. Just select your environment, and FUSION Profiler walks you step-by-step through your configuration. This means: ease of configuration, and a perfect build every time.***
***Using the FUSION Profiler enables almost anyone to configure these powerful FUSION applications for embedded systems.***

Pacific Softworks

*FOR MORE INFORMATION CALL:*
*(800) 541-9508 within USA*
*(805) 484-2128 outside USA*
*(805) 484-3929 FAX*
*Internet: sales@pacificsw.com*
*Europe, Middle East and Africa*
*Ph: (44) 1494-432-735*
*FAX:(44) 1494-432-728*
*World Wide Web: http://www.pacificsw.com*
*Distributors Worldwide*

CIRCLE # 45 ON READER SERVICE CARD

# What Have You Debugged For Me Lately?

by NICHOLAS CRAVOTTA

With increased integration of tools, debuggers have come to possess, perhaps indirectly, more and more functionality. Debuggers can control compilers, linkers, emulators, simulators, profilers, and a growing slew of other tools without bothering you for details. However, once you get past the issue of whether a particular debugger supports your processor of interest, the marketplace is a battlefield of feature-spawning monsters, ominous tools that can do just about anything you want them to. Yet, while these monsters have plenty of exciting and even grotesque features, they tend to have appendages that aren't particularly useful. A flipper is great if you need a monster to swim, but a prehensile hand is typically much more useful.

## THE HEART OF THE MATTER

If the debugger is the heart of your development system, the editor is the heart of your debugger. Brochures will tell you that a debugger is "easy to use," but under what circumstances is it easy to use? One way to understand "easy to use" is to focus on how you use your debugger most often. If it's difficult to control the features you use the most, you're liable to go crazy. Here are some considerations to keep in mind:

- Some editors automatically reformat your source code for you; some editors highlight keywords and others control tab spacing
- Editors can navigate code in many ways. For example, the editor may locate logical blocks, allowing you to click on an `IF` to find the corresponding `ELSE` and `ENDIF`
- If you've just spent half an hour opening all the right windows and placing them perfectly around the screen, you're definitely going to want the ability to record the setup for tomorrow's debugging session
- Assembly is sometimes displayed alongside C source code, sometimes interspersed between lines of code. If assembly is in between, you may have difficulty seeing more than a handful of source lines at one time
- Having the ability to disable the display of source code comments, especially the volumes produced by automatic code generators, can clear up a listing window
- Extensive on-line help can reduce the need for thick manuals on the desktop—especially if you can, for example, click on unfamiliar assembly instruction mnemonics and view a technical description
- The compiler knows all of the global and local variables accessible from a particular function. This means the debugger should be able to bring up a window of all variable names within the current scope, saving you from having to remember correct spellings or where an instance of the variable appears in the source code in order to open a watch window
- The size, type, and location of a variable can be as important as the value during debugging. Making the display of such information optional can keep watch windows compact
- "Out of scope" variables are often displayed as "not valid," taking up valuable screen space in the watch window. There are times, however, when you need to know the value of an "out of scope" variable, and you should not have to wait until the variable is back in scope to view its contents
- Being able to freeze the contents of a variable display window allows you to compare results to a later event without having to print the values to paper
- When debugging multiple tasks, the various windows associated with each task needed to be somehow grouped together, employing color-coding or some other scheme
- One-button system builds are a must, as well as are incremental compilation
- Downloading object files to your target should be simple. Incremental link-and-load support can reduce long downloads by downloading only changes

This is just a short list of some common features. You may want to confirm how they are implemented before committing yourself to tearing your hair out every day because the debugger does things the wrong way—that is, not your way.

## DEBUGGER LOYALTY

Depending on your project, you may begin development using a simulator, move to cross debugging (where the debugger is not on the target), and then employ an emulator. In some cases, you can use the same debugger in all three situations. Additionally, if you work across multiple platforms with different real-time operating systems (RTOSs),

# Software Debuggers

either on the same project or different ones, sometimes you can use the same debugger.

Some debugger vendors achieve this portability by conceptually dividing the debugger into two parts: the debugger environment, which is the same across the board, and a debugger interface, which is specific to a chip, RTOS, emulator, or monitor and across either serial, Ethernet, simulator data stream, or proprietary connection mechanisms. When you tell the debugger environment which of these you will be communicating with, the environment can dynamically customize itself, offering commands that apply to the particular context and tools. For example, a full emulator offers different functionality than background or JTAG emulation, and each RTOS offers different analysis support. Debuggers designed as open systems offer APIs capable of operating in a variety of applications and with many other tools. Some debugger interfaces even offer limited skeleton code for getting on-chip peripherals up and running so you can start code development before you've written any device drivers.

An important characteristic of each type of debugging is whether it is intrusive or not. Intrusive debugging means that in some way the debugging tools can affect the execution of an application. Many analysis features require some kind of instrumentation in your code that requires time to execute. For example, having a profiler instrument your code so that you can see what's happening can slow down execution. If you are running at peak performance, profiler overhead can cause side effects or failure that do not truly exist within your system.

## RTOS AWARENESS

A debugger that interfaces with an RTOS may offer RTOS-aware features. Such a debugger will recognize RTOS system objects, such as queues, and be able to track intertask communications.

On the highest level, RTOS awareness supports dynamic analysis tools for monitoring tasks. Time graphs can display important interrupt information such as when an interrupt request arrives and when servicing of the interrupt actually takes place. This feature is useful for discovering if an interrupt process takes place within a certain time frame (and is the interrupt therefore deterministic enough to process real-time data without loss?) or if priority inversion occurred (did a lower priority interrupt hold up a higher priority interrupt?). You can also determine peak performance requirements for the system when it is under stress and measure how long a particular function took to execute. Displays can show CPU logs, system logs, tasks in order of priority, data streams between tasks, and dynamic statistical data such as how many times a particular task executes. RTOS participation in analysis is less intrusive than instrumenting code because the overhead for analysis occurs during RTOS functions such as context-switching, not in the middle of a function in which interrupts might be disabled.

On a code level, an RTOS-aware debugger can ease the pain of debugging multitasking systems. A control window shows all threads running on a processor. You can suspend, spawn, or resume thread activities. Clicking on a thread brings up a debug window for that thread. Debugging message passing between tasks is easier because you can suspend the receiving task and trace the origination of the first task's message.

Setting breakpoints in a multitasking environment brings up the issue of whether a breakpoint is global or local to a single or defined group of tasks. For example, setting a breakpoint in a system-level function should only break in the tasks you want it to. There are two ways to implement this kind of breakpoint: the breakpoint is always set, and when it triggers, the debugger determines if this is a task for which it is enabled; or the RTOS sets the break-

# And some people still think Microtec is just a tools company.

**JIM READY**
**MICROTEC**

Embedded Software Expert

68K, CPU32, x86, PowerPC,™ ColdFire,™ ARM and i960 Supporter

Confidant

RTOS Guru

Networking Expert

Software Debugger

Legacy Software Migrator

Butt-Saver

HW/SW Design Authority

Telecom Specialist

Partner

JavaOS™ Licensee

C++ Wizard

Windows® Developer

Internet Enabler

But we're so much more. Our complete line of products and services can differentiate your designs, add value, and get you to market faster.

Products like Spectra®, Microtec's fully integrated—yet remarkably open—embedded development environment. It includes VRTX®, the industry's most powerful and proven RTOS, as well as our renowned XRAY® debugger and C/C++ compilers. Use them together, separately, or with your own choice of RTOS and tools.

Or let us do it for you. Microtec offers a full range of consulting and professional services, so we can provide everything from off-the-shelf point products to completely integrated, turnkey solutions.

So if you're ready to slice serious time and money off your next development project, call Microtec at **1-800-950-5554, Dept. 300**. Mention this ad, and we'll send you your own copy of the Embedded Software Expert badge.

Think of it as a little reminder from your total solutions company.

Microtec®
A Mentor Graphics Company

www.mri.com

**CIRCLE # 48 ON READER SERVICE CARD**

## OTHER ASPECTS OF DEBUGGING

Traditional debuggers cover the implementation phase of design and some in-field debugging. Yet debugging covers only a part of the territory you need to explore as you monitor a design to determine where and how it fails to meet expectations. Today, tools can help you verify your application on the specification and system levels, build a proof-of-concept, simulate and exercise an user interface, automate test suites, and manage software to avoid human error:

- **Specification and system level tools** allow you to use state machines or block diagrams to build a running model of your system and prove out a design. At this stage, you can verify interaction between system components, explore the effects of boundary and error conditions, and possibly create a virtual prototype of your design
- **Virtual prototypes** allow testing of user interfaces for usability and a determination of just how idiot-proof a system really is. Some virtual prototypes are executable files you can ship to the people who will eventually sign off on your design, giving them a chance to determine if you have met their expectations (not necessarily specifications) for the application early in the design cycle
- **Simulators** offer the opportunity to write and test code before hardware even exists. Simulators also provide an intimate window into a processor, allowing examination of cache performance and pipeline efficiency
- **Profilers** verify code coverage (whether or not every line of code executed during testing) and characterize where the majority of execution time is spent, thus locating bottlenecks and potential functions to hand-optimize
- **Test suites** automate the testing of a system. Test suites can test systems overnight, verifying that the latest set of changes has not introduced any old bugs back into the system by running test cases that expose all known bugs
- **Software management tools**, such as version control software, help manage code among teams of developers. These tools can prevent developers from unwittingly wiping out each other's changes. Some software management tools track all changes made to the system by developer, descriptions of the fixes, and reasons for the fixes, thus allowing other developers to know who wrote a particular section of code and why the program has that statement that looks like it does absolutely nothing useful at all

Traditionally, you might have mocked up a few of these tools yourself. For example, an algorithm could be proven using a BASIC program. The problem with designing your own tools is the time spent not designing your application; you'll spend valuable, the-project-is-late debugging time debugging your debugging tools. Many of today's debuggers, if they do not add functionality directly to the debugging environment, integrate directly with other tools, providing seamless access to test suites and profiling tools, for example. Remember, good design tools allow you to monitor your application and determine where it fails to meet your expectations. Many bugs arise from the development process itself through human error, and the right tools can save you hours of single-stepping by allowing these bugs to be identified in earlier stages of the development process.

point every time there is a context switch to a task for which it is enabled and resets it when leaving the task. The first method can hiccup a system because breakpoints will trigger during tasks in which they may not be desirable. The second method adds minor overhead to the context switching process, depending on how many breakpoints you have activated.

Multiprocessor debugging is similar to multitasking debugging except that each processor needs to have an RTOS or monitor through which the debugger can communicate. This boils down to an interface issue as to which processor is the control processor that the debugger hooks into, and which processors communicate debug information through that control processor.

Finally, on the lowest level, an RTOS-aware debugger can help integrate an application on top of an RTOS, giving you the perspective of the RTOS as part of the application. This can be useful while developing device drivers or chasing problems that trash the RTOS itself (or the debug monitor). If you have a proprietary RTOS and the debugger has an open API, you may be able to map your proprietary API to the debugger API and observe your RTOS in motion.

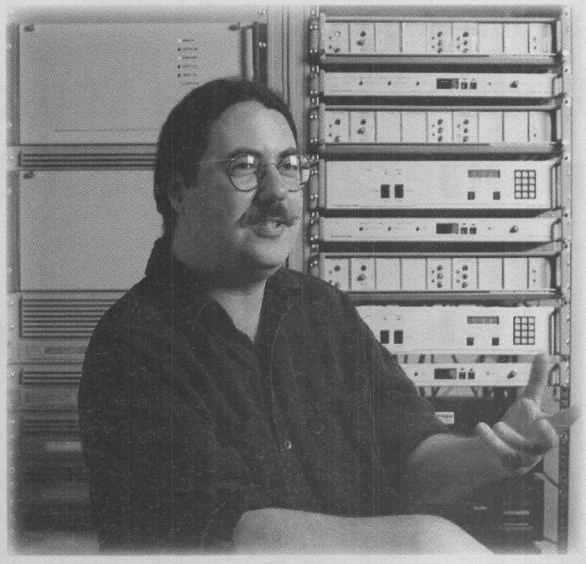
### ADVANCED SUPPORT

Some less used advanced features of interest include profiling, version control support, and memory error detection. (See "Other Aspects of Debugging" for a description of these and other advanced debugging tools.) Here are a number of ways to instrument code for profiling:

- Single-step through code
- Use an auxiliary timer and interrupt to record the instruction pointer $x$ times per second
- Record an instruction by instruction log from a simulator
- Collect trace data from an emulator

By running profiles when the application is in different configurations, you can build a typical picture of which part of your code runs most often. Note that the first two methods are intrusive. The third method is not intrusive but it is also not real-time. The fourth method is both non-intrusive and real-time.

The second method above, using an



"The Diab Data compiler is simply the best embedded compiler around. We did an evaluation of various C++ compilers for the PowerPC, and the Diab Data compiler produced the tightest, fastest code. In the last several years, I've occasionally re-evaluated the Diab Data compiler. It's still the best. Diab Data definitely lives up to its reputation for technical excellence."

**Jim Houha, Kernel Engineer**
**PowerTV, Inc.**
**Set-Top Box Innovator**

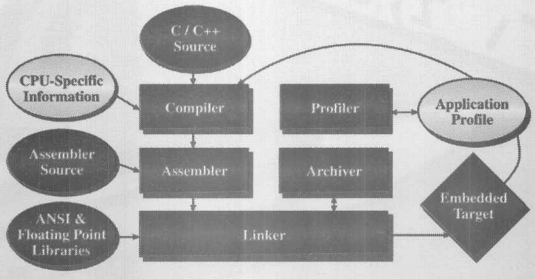
# The Expert's Choice!

Diab Data's highly optimizing C & C++ compiler suites are the expert's choice for advanced embedded applications. When you need the fastest, tightest, most reliable code possible, Diab Data's Power Compiling Solutions™ deliver.

## The Performance Leader

Diab Data's D-CC and D-C++ compiler suites set the performance standard for fast, compact, high-quality code. With both CPU-specific as well as application specific optimizations, D-CC and D-C++ generate the optimum code for enabling faster, more competitive applications.

**CPU and Application Specific Optimizations Generate the Optimum Code for Your Design**




## Superior Programming Flexibility

Diab Data's Power Compiling Solutions offer superior programming flexibility. With precise control over compiler, optimization and linker options, you can tailor programs for the extreme design constraints of today's high-performance embedded designs.

## Open Solutions

Diab Data's standards compliance and open environment philosophy allow you to choose from an unparalleled selection of debug and run-time environments with proven tool interoperability.

## The Motorola RISC Leader

Diab Data is the market leader for embedded PowerPC and ColdFire applications.

Come join the growing number of expert developers who enjoy the superior performance, reliability and responsive service provided by Diab Data.

***Diab Data's Power Compiling Solutions. The Expert's Choice. For Experts Like You.***

MOTOROLA EMBEDDED PLATINUM DEVELOPER

DIAB DATA
*Defining Compiler Performance*

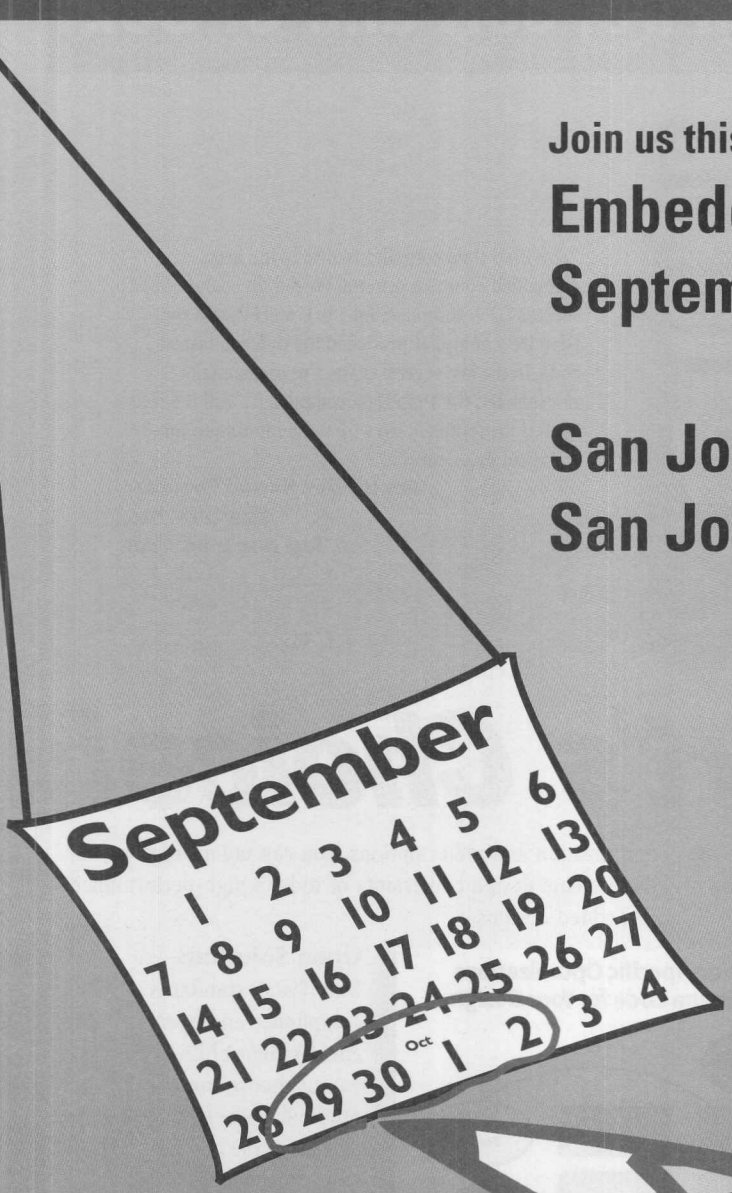
**Browse Us**
**www.ddi.com**

**E-mail Us**
**info@ddi.com**

**CIRCLE # 49 ON READER SERVICE CARD**

Diab Data Headquarters: Tel: 415-571-1700, Fax: 415-571-9068. Central Region: Tel: 630-529-6950. Eastern Region: Tel: 215-362-1786.
EUROPE: Diab Data GmbH, Tel: +49 (0)89-9393-1191, Fax: +49 (0)89-930-5184

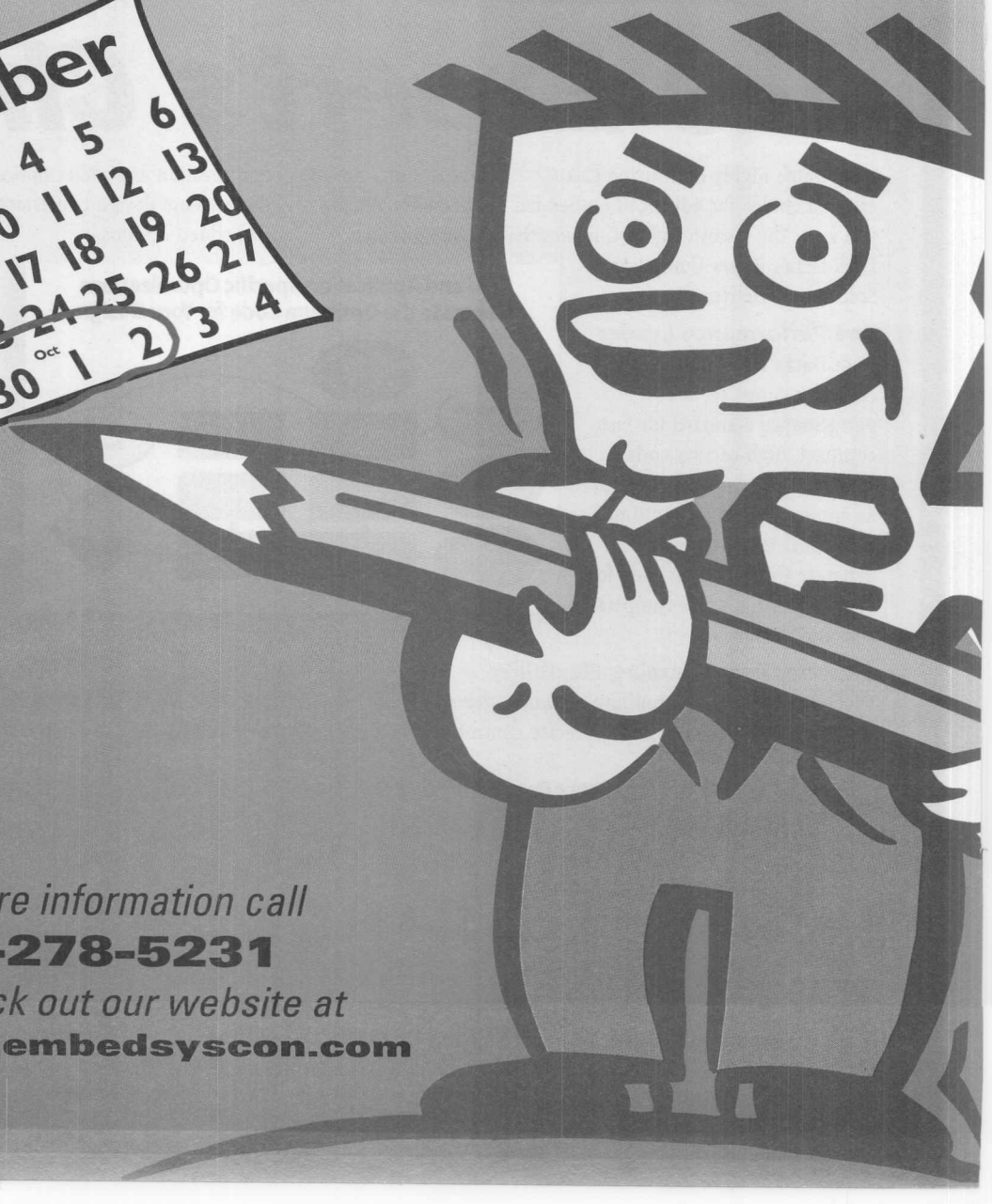
# Mark Your Calendar




**Join us this fall for the**

## Embedded Systems Conference
## September 29 - October 2, 1997

## San Jose Convention Center
## San Jose, CA



EMBEDDED SYSTEMS CONFERENCE

*For more information call*

**415-278-5231**

*or check out our website at*

**www.embedsyscon.com**

auxiliary timer, actually makes a statistical reckoning of which code consumes the most execution time. Even with a couple of thousand samples, you can get a realistic idea of where your bottlenecks are. Despite the fact that your application may have a million lines of code, only a small subset of this code will execute regularly, and chances are a random sample will frequently fall within it.

Some debuggers have integrated with version control tools. When you open a window to debug code, the debugger checks out a read-only copy of the source. When you begin to make changes, the debugger will attempt to raise your rights to write access. The advantage of version control software is that if a section of undocumented code makes no sense or appears to be in error, you can call up its history to trace the changes the code has gone through and who made them. You may be able to get a comparison listing of the code against the last set of changes. In any case, you can get a better idea of what the code is supposed to do or what bug it fixes instead of simply having to guess or curse out the idiot responsible for the code (who might even turn out to be yourself!).

The main problem with version control software, however, is that you don't want to make experimental changes to code that others may count on as reliable. In this case, you'll need to create a experimental directory branch, separate from the original source files but initially a perfect duplicate, for building and debugging your application. Create another duplicate of the master directory so that when your experiments are over, you have an original copy to compare against—the master directory may have been changed by someone else by then.

Memory error detection software can help you quickly find common types of errors associated with allocating and using memory. These tools are highly intrusive, instrumenting your code to detect use of uninitialized

# Software Debuggers

pointers, memory leaks, and other common memory misuses. When an error is detected, the program breaks and notifies the debugger of the error, offering information such as the origin of a particular allocation and where the error actually took place. Make sure that you have access to a list of calling functions that goes back a number of levels. Often these errors will be caught in `malloc`, and if you have a wrapper around `malloc`, you need to know who called the wrapper to determine the leak's location.

## COMMAND LINES

Almost everyone is pushing graphical user interfaces (GUI) these days. GUIs are more intuitive, easier to use, and better for you than command lines interfaces. I've never been convinced of the absolute truth value of this last statement, having grown up on command line interfaces. I would prefer to see a hybrid approach offering both interfaces, for each has powerful advantages. For example, with a command line, your debugger can record a series of commands and repeat them or allow you to alter a single value and run the whole sequence again. This is not as easily accomplished when commands are mouse clicks, although macro tools are improving. More importantly, however, you can display a list of previously entered commands and understand what you just did wrong. Displaying the last fifty mouse clicks (oh, were you recording them?) yields a sequence of text commands conceptually different from the mouse clicks associated with them.

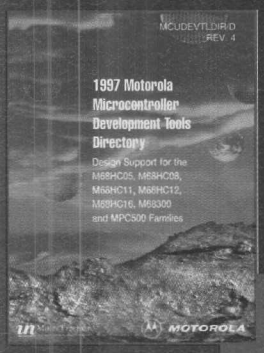
Having an active command line is critical. You could zero out all the values in an array one at a time using a dialogue box, or you could write a quick loop on a command line to do it for you. While stopped at a breakpoint, you can copy lines of source code to the command line and execute them out of order, even calling functions or library routines. You can display data or analyze it without altering the data or leaving the environment. Traditional debugging environments require you to capture data to a file, then run your home grown display program outside of the environment to view the results.

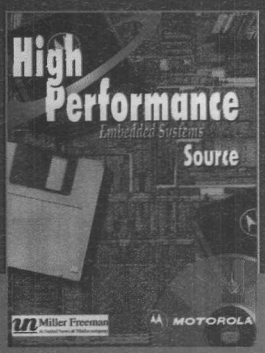
Many debuggers are beginning to feature application-specific and true object-oriented debugging. For example, 1,000 points of waveform data make much more sense when displayed as a waveform and not as 1,000 contiguous words of memory. You can create your own displays with buttons and parameters to display data in its most useful formats. For objects,



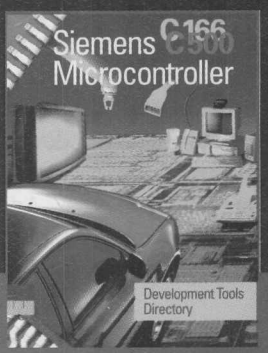
# Miller Freeman Directories

## Your One-Stop Information Source





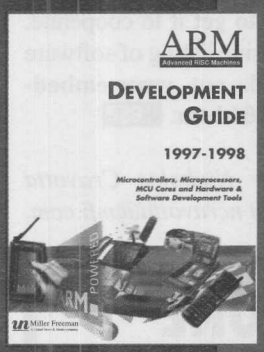




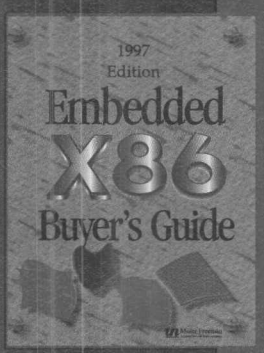
*When making decisions about your hardware and software tools . . .*

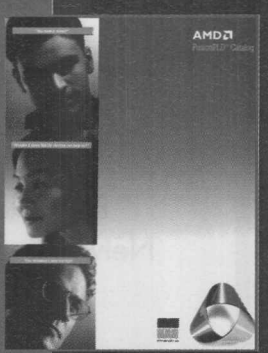
**Miller Freeman**
**Product Directories**
*provide you with the critical data you need*

**For your FREE directory**
**call +1-800-500-6815**
**or +1-913-841-1631**







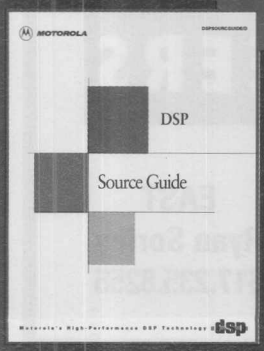



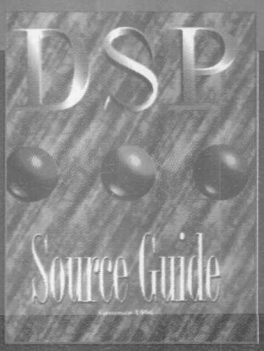

Miller Freeman
A United News & Media company

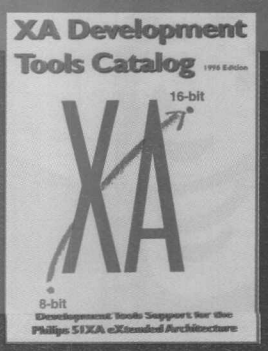
EMBEDDED SYSTEMS CONFERENCE

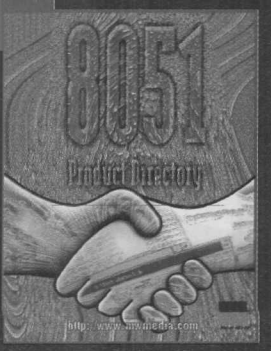
Embedded Systems PROGRAMMING

**www.embedded.com**



debuggers should allow you to have specific debug object extensions, not necessarily intended for download to a target, for displaying and analyzing data in a manner natural to the object. To display a linked list, for example, you should not have to follow the NEXT or PREV pointer 10 times to see the tenth item in the list. A window displaying the five previous items and the next five items could save you from having to open 10 variables and arrange them on the screen.

## DEBUGGING THE DEVELOPMENT PROCESS

Application development methods have changed and continue to change. With memory prices dropping, writing compact code isn't as important as getting the design to market. Faster processors transfigure sloppy code that was slow a year ago into functional and efficient code today. Upgrading a processor or clock, while throwing money at the slow software problem, may save you money in the long run by reducing development times and avoiding challenging code optimization. In general, embedded development doesn't involve byte or cycle counting the way it used to.

Additionally, as processors become more complex, traditional methods fail to do the job. For example, it can be difficult to understand exactly what a processor is doing as caches grow and pipelines become longer and wider. Visualization of objects becomes more complex as the objects themselves become more complex. Applications such as signal processing call for standard display options of data, including graphical plots and autoscaling.

And so we come full circle. As applications become more complex, design tools need to simplify that complexity. In one sense, the fact that your debugger has a super feature that you might possibly use once is relatively unimportant. If it's a worthwhile feature, the other debuggers will pick it up within six months to a year. What matters is what your debugger does for you today and every day, and that it does it the way you find most useful. Just as a small portion of your code constitutes the majority of execution time, you'll find yourself using mostly a small subset of the debugger's feature set—a subset that, hopefully, doesn't make you hate your debugger or spend all of your time trying to get it to cooperate. For a comprehensive listing of software debuggers, check out **www.embedded.com/97/sr9706.htm**. ESP

*Technical editor Nicholas Cravotta can be reached at ncravotta@mfi.com.*

*by Don Morgan*

# Fundamentals of FIR Design

Sometimes the math involved in signal processing obscures the underlying simplicities. Math, like it or not, is the most descriptive language available in these matters. Unfortunately, despite the discipline inherent in mathematics, it is not a standardized subject—each author uses his or her favorite notations for whatever is being discussed. It isn't uncommon to be unable to find the definition of a particular symbol in a particular discussion or derivation.

This column marks the beginning of a short study of the elements of digital filter design, beginning with FIR filters. In addition to the mathematics, I will also introduce some code as an example of applying the techniques involved. The principles involved are based upon certain dualities in signal processing that can make the subject more tractable for those who aren't as well-versed in the subject.

**The FIR filter provides some distinct advantages, including inherent stability and ease of design.**

## TIME DOMAIN/FREQUENCY DOMAIN

The FIR filter, a filter without poles, is an invention of digital signal processing—it doesn't even exist in analog signal processing. The filter is actually a polynomial approximation following directly from the Weierstrass theorem, which states that for any function $f(x)$ continuous on $[a,b]$, there exists a sequence of ordinary polynomials which converges uniformly to $f(x)$ on $[a,b]$. Loosely, this means that given a sufficient number of terms, we can usually approximate any function that exists in nature with a polynomial.

FIR filters can be designed to represent any magnitude response the designer chooses. But they fall prey to the same problems that plague anyone who attempts a polynomial approximation—quite a few terms may be needed to replicate every wiggle in $f(x)$. Nonetheless, the FIR filter provides some distinct advantages, including inherent stability (because it's bounded by definition), ease of design, linear phase, and a group delay that depends upon filter length and symmetry.

## THE DIRECT APPROACH

Illustrations of spectra are often used to explain how a filter works. Figure 1 illustrates the frequency response of a low-pass filter that we might like to design.

How can we derive filter coefficients that will produce those results? Actually, it isn't as difficult as it may seem. We need to remember one of the most important dualities in signal processing, in which time domain impulse response and system (frequency) response form a Fourier transform pair. In other words, the Fourier transform of the impulse response of a system is the frequency response and conversely, the Fourier transform of the frequency response is the impulse response. If we have one, we can easily get the other by applying the transform.

Figure 1 is an illustration of the frequency response of a proposed FIR low-pass filter. To get the impulse response (the filter coefficients), we simply need to perform a Fourier transform on the data in the illustration.

Unfortunately, you cannot collect the data from a graph of magnitude response, because these graphs are almost always of *squared* magnitude response, or the result of a transform in which the absolute value of the magnitude was taken for the plot. We cannot know whether a particular value is positive or negative by looking at such a graph. But let's pretend for the moment that you have access to the values used to prepare the plot (as I do), and we can collect the data from the graph. See Table 1.

The plot in Figure 1 was actually created by taking the absolute value of the data in Table 1. With this information, we can take the inverse transform of these magnitudes to obtain our filter coefficients. Figure 2 presents a plot of the result of this inverse transform. This plot is the impulse response of the proposed filter and a table of the filter coefficients (Table 2).

Figure 2 is a plot of a function of the filter coefficients derived from the inverse transform. Some will recognize it as a function that we have mentioned before and will run into time and time again in filter design—the sin*c* function. Mathematically, it is defined:

$$\sin c\, x = \frac{\sin \pi x}{\pi x}$$

Plotting a simple graph of this function reveals that it possesses the following properties:

$$\sin c\, 0 = 1$$

$$\sin c\, n = 0$$

and:

$$\int_{-\infty}^{\infty} \sin c\, x\, dx = 1$$

**FIGURE 1**
*Illustration of the frequency response of a desired low-pass filter.*

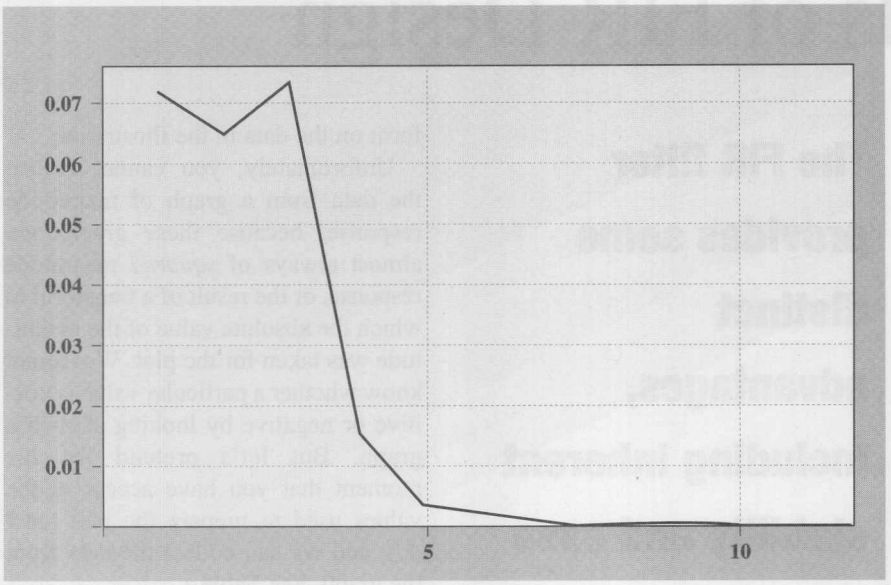



**TABLE 1**
*Frequency response of desired filter.*

| |
|---|
| 0.0743541 |
| -0.0671585 |
| 0.0745791 |
| -0.0167574 |
| -0.00349554 |
| -0.0015938 |
| -0.000907476 |
| -0.000578444 |
| -0.000399716 |
| -0.000299329 |
| -0.000247441 |
| -0.00023134 |
| -0.000247441 |
| -0.000299329 |
| -0.000399716 |
| -0.000578444 |
| -0.000907476 |
| -0.0015938 |
| -0.00349554 |
| -0.0167574 |
| 0.0745791 |
| -0.0671585 |

The sin*c* function possesses the interesting spectral character of containing all frequencies up to a certain limit and none above. You may recall from previous discussions that this is similar to the sampling function used to determine the impulse response of a system. The narrower the function is, the higher the frequency components it can measure. In the illustration, note that the general appearance of the sin*c* function is similar to an impulse.

The sin*c* function forms a transform pair with the box function in the frequency domain. Essentially, our low-pass filter is a box whose origin coincides with the origin of the spectrum. We call this an ideal low-pass filter. It expects minimal deviation in passband and minimal deviation in the stopband and a sharp transition band—thus creating the box.

But let's not obscure one of the most important results of this transformation and discussion. The formula for the sin*c* function provides us with a closed form for the calculation of filter coefficients from known parameters. We do not need to pursue the transforms for our answers; we need only know how many coefficients we want and what our cutoff and sampling frequencies are, and we can solve for the coefficients. This is definitely one of the perks associated with a familiarity with the Laplace and Fourier transforms.

## DESIGN

When we design a digital filter with known parameters, we are defining an essentially infinite length impulse response with those parameters. When we use the Fourier series to design FIR filters, we are actually performing a least-squares fit in the frequency domain to a particular magnitude response, as we demonstrated at the beginning of this column.

For the purpose of simplicity, we can state (for the time being) that the more coefficients, the better the filter approximation (as with Weierstrass). But because we can't support an infinite number of coefficients, we must truncate somewhere, giving us the number of coefficients. We'll also make a stipulation that we'll discuss in more detail later: these coefficients are symmetrical about a central ordinate that we calculate immediately, that is, the direct proportion of the cutoff frequency of the low-pass filter and the Nyquist frequency.

We can determine the coefficients that will express this impulse response by integrating:

$$c(n) = \frac{1}{2}\int_{-\pi}^{\pi} A(e^{j\Omega}) e^{j\Omega n} d\Omega \qquad (1)$$

This is the kernel expression for the Fourier series. We are solving for the Fourier coefficients that will allow us to expand this magnitude response in a Fourier series. The Fourier series can be shown to produce the best least squares approximation in a Hilbert space.

Because it is an approximation, we also have an error term which represents the difference between the actual continuous time function and the digital representation. Let us show this as:

$$A(e^{j\Omega}) - A[e^{j\Omega}] \Leftrightarrow a(n) - a[n]$$

Let's call this error term $\varepsilon$, and define it as:

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \left| A(e^{j\Omega}) - A[e^{j\Omega}] \right|^2 d\Omega = \varepsilon$$

and:

$$\sum_{n=-\infty}^{\infty}\left|a(n)-a[n]\right|^2=\varepsilon$$

The above is given here as signal energy in the form of Parseval's Theorem.

As the error is minimized by using the orthonormal spectral coefficients produced by the integral (see Equation 1), the error term is orthogonal to the approximation. In fact, they form a direct sum, and because the error would be zero if the approximation used all basis functions (as, if it were infinite, the integral would), it stands to reason that the more basis functions used, the smaller the error term will be. Remember our previous statement: the more terms we use, the better the approximation.

The ideal low-pass filter has a frequency amplitude response, as follows:

$$A\left(e^{j\Omega}\right)\begin{cases}1, & for\ |\Omega|<\Omega_c\\ 0, & for\ \Omega_c<|\Omega|\le\pi\end{cases}$$

Using the formula, we can calculate the necessary coefficients for our low-pass filter. How do we do this? Let's start with an abbreviated derivation. We will use the Fourier series to approximate the desired magnitude response:

$$a(n)=\frac{\sin(n\Omega_c)}{\pi n}$$

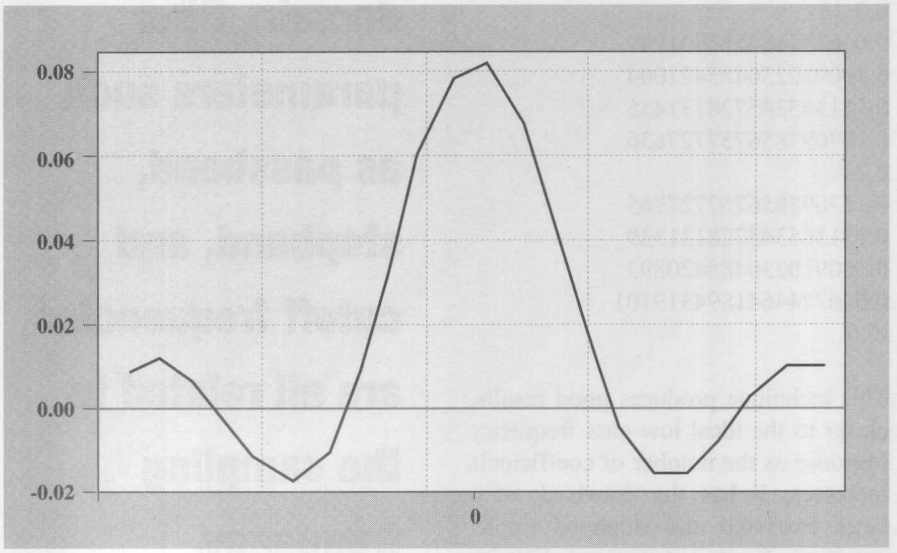
with

$$\Omega_c=\frac{2\pi f_c}{f_s}$$

where $f_S$ is the sampling rate. The difference between this representation of the sinc function and the one presented earlier is that $X$ has been replaced with $\Omega_C$.

I should offer a word of caution. What we derive with these coefficients is a non-causal filter. The sinc function, as you will note from Figure 2, is symmetrical around zero, which means

**FIGURE 2**
*Plot of the impulse response of the filter characteristic in Figure 1, with the filter coefficients.*




that we expect some output from our filter before there is an input. We must move the origin of our filter coefficients so that they all occur after time = 0. The causal impulse response will be a simple shift in time:

$$h(n)=a\left(n-\frac{N-1}{2}\right),\ n=0,1,2,3\ldots N-1$$

We can find filter coefficients for other forms of ideal filters as well, using these formula:

highpass:

$$a(n)=-\frac{\sin(n\Omega_c)}{\pi n}$$

bandpass:

$$a(n)=\frac{\sin(n\Omega_{ch})-\sin(n\Omega_{cl})}{\pi n}$$

bandstop:

$$a(n)=\frac{\sin(n\Omega_{cl})-\sin(n\Omega_{ch})}{\pi n}$$

## USING THE FOURIER METHOD

In the digital domain, filter parameters such as passband, stopband, and cutoff frequencies are all related to the sampling frequency. In fact, they form a proportion. Let's say that we wish to design a simple low-pass filter with a cutoff frequency of 10KHz ($f_S$ = 10KHz). This number means nothing without the sampling frequency, which in this case will be 48KHz ($f_S$ = 48KHz), making the Nyquist frequen-

**TABLE 2**
*Figure 2 filter coefficients.*

| |
|---|
| 0.00861451 |
| 0.0106022 |
| 0.00612134 |
| -0.00358194 |
| -0.0137832 |
| -0.0180861 |
| -0.0113682 |
| 0.00748951 |
| 0.0344581 |
| 0.0614927 |
| 0.0795775 |
| 0.0823817 |
| 0.0689161 |
| 0.0439234 |
| 0.0159155 |
| -0.00633728 |
| -0.017229 |
| -0.0161823 |
| -0.00723432 |
| 0.00329539 |
| 0.00984516 |
| 0.00991819 |

cy 24KHz. We choose a value for the number of coefficients and calculate:

```
0.0
0.046774464189431999
0.100910230485421004
0.151365345728131455
0.187097856757727836
0.2
0.187097856757727765
0.151365345728131339
0.100910230485420893
0.046774464189431901
-0.0
```

This technique produces good results, closer to the ideal low-pass frequency response as the number of coefficients increases. It has the drawback of a large passband and stopband ripple, resulting from a truncated Fourier approximation. The drawback is known as Gibb's phenomenon and can be solved (or at least ameliorated) with a number of techniques, including windowing and finite transition banding. We will pursue those subjects in future columns, in addition to other closed and open forms of FIR design such as Parks-McClellan and the Lagrange approximation.

**In the digital domain, filter parameters such as passband, stopband, and cutoff frequencies are all related to the sampling frequency.**

## AS A PROGRAM

Now let's look at what is required to write a simple program to solve for the coefficients we've described. The listing I've provided presents a short bit of code meant only to demonstrate the process involved. Much of the accompanying program (available from http://www.embedded.com/code/htm) is involved with file handling and some mild error detection. You will need to provide certain command line parameters to the routine so that it can solve for the coefficients. If you don't, a short screen will remind you of the necessary parameters. Very little error correction is performed on the data you provide.

There are a few variables that are important to the computation of the filter coefficients. The variable `coef` defines the number of filter coefficients; `lcutoff` sets the cutoff or corner frequency for the low-pass and high-pass filters. Finally, `hcutoff` is used in the bandpass and bandstop filters for the upper cutoff point. Note that `lcutoff` and `hcutoff` are normalized to Nyquist. (This simple divide represents the integration of the function from zero to $\pi$.)

Next, the core of the program is given in the case statement:

```
switch(lambda) {
  case 0: //LOWPASS
    vector[coef/2] = hcutoff;
    //central coeficient is normalized coef
    for(i = 1; i<coef/2; i++) {
    //write both halves of symmetrical filter
    //at the same time
      vector[coef/2+i] = vector[coef/2-i] =
        sin(vector[coef/2]*i*PI)/(i*PI);
      }
  case 1: //HIGHPASS
    vector[coef/2] = hcutoff-lcutoff;
    //central coeficient is normalized coef
    for(i = 1; i<coef/2; i++) {
    //write both halves of symmetrical filter
    //at the same time
      vector[coef/2+i] = vector[coef/2-i] =
        (sin(hcutoff*i*PI)-
        sin(lcutoff*i*PI))/(i*PI);
      }
    break;
  case 2: //BANDPASS
    vector[coef/2] = 1-(lcutoff);
    //central coeficient is normalized coef
    for(i = 1; i<coef/2; i++) {
    //write both halves of symmetrical filter
    //at the same time
      vector[coef/2+i] = vector[coef/2-i] =
        -1*fabs(sin(vector[coef/2]*i*PI)
        /(i*PI));
      }
    break;
  case 3: //BANDSTOP
    vector[coef/2] = 1-(hcutoff-lcutoff);
    //central coeficient is normalized coef
    for(i = 1; i<coef/2; i++) {
    //write both halves of symmetrical filter
    //at the same time
      vector[coef/2+i] = vector[coef/2-i] =
        (sin(lcutoff*i*PI)-
        sin(hcutoff*i*PI))/(i*PI);
      }
    break;
  default: //WHO KNOWS??
    printf("\ndon't know that filter
      function...");
    exit(3);
}
```

The variable `lambda` is a value that selects the proper coefficient derivation scheme, `coef` is the number of desired coefficients, and `vector` is the output. The program produces the coefficients distributed correctly for an odd-length symmetrical filter.

## MORE TO COME

Next month we'll discuss even and odd filter lengths and the difference between them. We will also introduce another closed form for developing filters (the Lagrange approximation) as well as some code for using the coefficients you develop. ESP

*Don Morgan is senior engineer at Ultra Stereo Labs and a consultant in signal processing, embedded systems, hardware, and software. Morgan is currently completing a book about numerical methods, featuring multirate signal processing. He is also the author of* Practical DSP Modeling, Techniques, and Program-ming in C *(John Wiley & Sons, Chichester, U.K.).*

New Release! Now Shipping!

# Nine years of Embedded Systems PROGRAMMING on one CD!


Release 2.0 includes the entire history of *Embedded Systems Programming* from 1988-1996 and the premiere 1996 Buyer's Guide issue with information on over **1,100** products. Over 98 issues **including full text, source code, buyer's guides, and product demonstrations** not found in the magazine are packed on one CD. All of your favorite columnists including Jack Crenshaw, P.J. Plauger, Jack Ganssle, Tyler Sperry, and Bruce Eckel are included. You now have the ultimate embedded development resource at your fingertips for **just $79.95!**

Buyer's Guides
articles
SOURCE CODE
Product Demonstrations

## Special Features!

- ✔ Powerful text and Boolean search engine
- ✔ Over 800 articles, reviews, and editorials
- ✔ Buyer's Guide information on 1,100 products spanning 16 product categories
- ✔ Copy valuable programming code directly into your designs
- ✔ Access to over a million words
- ✔ Product demonstrations and links to the Internet
- ✔ Copy, bookmark and notes functions
- ✔ Windows/Mac/Unix-based CD-Rom with a robust search engine

## The Ultimate Time Saver!

Imagine you are having trouble utilizing a processor's memory management unit (MMU) in your embedded design. Just put the Embedded Systems Programming Library into your CD-Rom, launch a search for "MMU", and presto, a slew of articles with valuable programming code on the subject. An intuitive user interface and potent search engine guides you in finding the information you need. Access nine years of back issues by title, author, month, year, or use a custom word search to locate exactly what you need. Jump directly to the information you want, then cut and paste what you need to your favorite word processor or code editor.

## Order Today!

**1-800-444-4881**
**Fax: 913-841-2624**

**E-mail: orders@mfi.com**
**Mail: Miller Freeman, Inc.**
**1601 West 23rd Street Suite 200**
**Lawrence, KS 66046-2700**
**Attn: CD Rom Order Processing**

$29.95 upgrade available to prior owners

## I want this CD!

*only $79.95 (+ shipping and handling*)*

Name (Please print clearly)

Address

City/State/Zip/Country

Phone email

Credit Card Number Exp. Date

Signature required

Check one:
☐ Visa/MC ☐ Amex ☐ Check enclosed

Check one:
☐ 1996 CD-Rom ($79.95 + shipping)
☐ Upgrade from previous edition ($29.95 + shipping)

*Shipping is $3.50 US/Canada; $15.00 all other countries. California ($6.80), New York ($6.60), Texas ($6.60), Illinois ($5.00), Kansas ($5.50), and Georgia ($4.80) residents should add sales tax.

Clip out or photocopy this form.

ESP3/97

# Tools For Embedded Developers

## Software Tools

### Micro Web Server Solution

Based on Internet standards and Java technology, Embedded Micro Interface Technology (EMIT) enables the use of a Web browser to configure and manage electronic devices over the 'Net or a direct connection. It is the first scalable micro Web server solution that makes this technology available to even the smallest electronic devices, such as the 8051. EMIT requires only 750 bytes of ROM and 28 bytes of RAM on the device, and includes pre-written software for most common device controls. A single-user EMIT 1.0 Software Developers Kit with 10 embedded licenses is $1,200. A company site license with 10 embedded licenses is $3,000.

EMIT 1.0
emWare
Midvale, UT
(801) 256-3883
**READER SERVICE NO. 115**

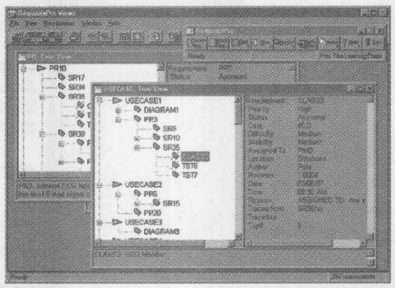
### OO Development Tool

ObjecTime Developer 5.0 is an object-oriented (OO) toolset designed for management of the entire development cycle. The OO toolset enables developers to build graphical models of their designs and observe these models on development and target platforms. The executable modeling capability aids in the development of real-time designs, as the graphical models are animated at every stage of the development life cycle.

ObjecTime Developer 5.0 adds code-generation support for Windows NT 4.0 and Wind River's Tornado for PowerPC and 68K, and now supports over 30 target combinations. A floating license is $25,000.

ObjecTime Developer 5.0
ObjecTime Ltd.
Kanata, Ontario
(613) 591-3535
**READER SERVICE NO. 116**

### Management Tool



The RequisitePro 2.5 requirements management tool adds integration with the Rational Rose visual modeling tool, and Microsoft Visual SourceSafe, a project-oriented version control tool. This integration makes it possible to trace end-user requirements from their origin through implementation and to manage changing requirements along with changes to the software source code. RequisitePro integrates into the leading application development tools used to define, model, manage, and test software. A single workstation license is $1,295. Concurrent licensing is available in quantities of 10 or more starting at $2,995 per license. The upgrade is free to RequisitePro customers with SupportPlus.

RequisitePro 2.5
Rational Software Corp.
Boulder, CO
(303) 444-3464
**READER SERVICE NO. 117**

### Development Environment

The MULTI Development Environment is now available for the CHORUS/ClassiX real-time operating system from Chorus Systems. Green Hills development tools, including C and C++ optimizing compilers, tool chains, utility programs, and the MULTI Development Environment, can be used to develop and debug multi-threaded applications supported by CHORUS/ClassiX from application code level to kernel execution level.

Microprocessor families currently supported include the 68K, PowerPC, and $\mu$SPARC II. The package, including a C compiler and development environment supporting CHORUS/ClassiX on a Unix host, is $7,900.

MULTI Development Environment
Green Hills Software
Santa Barbara, CA
(800) 500-2580
**READER SERVICE NO. 118**

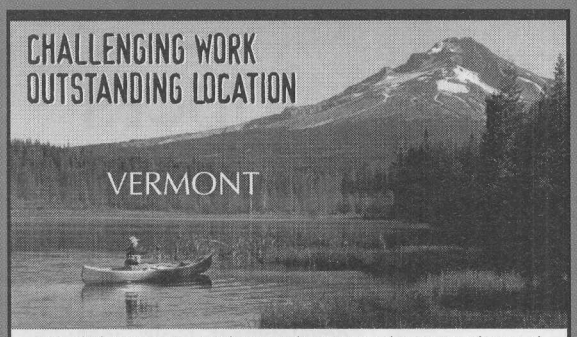
## Chips

### USB 8-Bit Microcontrollers

New additions to the 68HC05 family include the 68HC705JB2 and 68HC05JB2 microcontrollers for low-speed Universal Serial Bus (USB) computer mouse applications. Each chip has an on-chip memory of 2,048 bytes of user ROM (or EPROM) and 128 bytes of user RAM. The device also offers 16-bit input capture/output compare timers and is designed to comply with low-speed USB with three endpoints. The 68HC705JB2 (EPROM) with voltage regulator is $3.50 for pilot production volumes. The ROM-based 68HC05JB2 is $1.40 in high volume.

68HC705JB2/68HC05JB2
Motorola CSIC Division
Austin, TX
(800) 765-7795 ext. 887
**READER SERVICE NO. 119**

# EMBEDDED CAREERS




BFGoodrich Aerospace - Aircraft Integrated Systems is a dynamic manufacturer of state-of-the-art electronic performance systems for the aerospace industry. Business growth and internal promotions have created excellent opportunities for **Software Engineering Professionals** at our Vermont location.

Located near Burlington - ranked #6 by *Reader's Digest*'s "50 Best Places to Raise a Family," and cited as one of the 7 "Dream Towns" by *Outside* Magazine - our area offers outstanding recreational opportunities and much more.

## SOFTWARE ENGINEERS
### Embedded Systems

Design, code, test and integrate real-time embedded software and conduct design reviews. Programming languages include C, C++ and Ada (Ada training will be provided if necessary). Experience with large project design, CASE, OOA/OOD, and software reuse is preferred.

BFGoodrich Aerospace offers competitive salaries and a comprehensive benefits package. Please mail, fax or e-mail your resume to:

**Employment Manager**
**BFGoodrich Aerospace**
**Dept. EE**
**100 Panton Rd.**
**Vergennes, VT 05491**
**Fax:(802)877-4111**
**E-mail: bfgempl@aisvt.bfg.com**

**BFGoodrich**
Aerospace
Vergennes, VT

**BFGoodrich Aerospace is an equal opportunity employer m/f/d/v.**

---

**TRW Automotive Electronics**
*Farmington Hills, Michigan*

### Embedded Systems Opportunities

**Put your energy and creativity to work** for the leader in state-of-the-art automotive electronics R&D and manufacturing. **TRW** seeks individuals who desire a broad scope of responsibility, career growth opportunity and a dynamic environment.

- **Project Managers**
- **Software Engineers**
- **Hardware Engineers**
- **Individual Contributors**

Develop systems for state-of-the-art crash control, electronic steering, and convenience systems. Desired experience includes C coding, and assembly, algorithm development, embedded microcontrollers, and sensor systems. BS EE/MS EE and 2 or more years experience preferred.

TRW/AEG offers Fortune 100 level compensation and benefits and excellent relocation assistance. Respond to **TRW; c/o Myers & Associates; P.O. Box 238; Allen TX 75013-0238; Fax 972-390-1329; E-mail TRWJOBS@aol.com.**

★ Check out our Web site at **WWW.ALPHA-USA.COM/TRW-JOBS.**

An Equal Opportunity Employer, M/F/D/V.

TRW

---

# embedded.com

**has expanded to include**

**Embedded Systems PROGRAMMING**

**Miller Freeman Directories**

**Embedded Systems PROGRAMMING EUROPE**

**EMBEDDED SYSTEMS CONFERENCE EAST**

**EMBEDDED SYSTEMS CONFERENCE**

**Please visit! For advertising opportunities, call Marcy Walpert at 415.905.2348.**

# EMBEDDED MARKETPLACE

To find out more about the products and services displayed here, circle the corresponding Reader Service number on the response card bound into this issue. Each advertiser will send you more information free of charge!

Embedded Marketplace is a special showcase section reserved for advertisers with standard 1/9-page display ads (2 1/4 by 2 inches). For information on placing an ad in Embedded Marketplace, call Robin Lander (415) 278-5274 or Ryan Sorley (617) 235-8255.

# Advertiser Index

| ADVERTISER | PAGE | R.S. # |
|---|---|---|
| Accelerated Technology | 65 | 37 |
| Adtron Corp. | 86 | 54 |
| Advanced Transdata Corp. | 102 | 90 |
| Advantech | 80 | 47 |
| Advin Systems Inc. | 99 | 66 |
| American Arium | 101 | 80 |
| American Arium | 49 | 102 |
| Applied Microsystems Corp. | 39 | 20 |
| Archimedes Software | 68 | 101 |
| Atlas Computer Equipment Corp. | 103 | 97 |
| Atmel Corp. | 17 | * |
| Avnet Computer | 102 | 91 |
| Avocet Systems Inc. | 43 | 22 |
| Bagotronix | 103 | 96 |
| Beacon Development Tools | 18-19 | 10 |
| BF Goodrich | 98 | |
| BP Microsystems | C2 | 1 |
| Byte Bos | 102 | 89 |
| Byte Craft Ltd. | 110 | 58 |
| Cactus Logic | 102 | 88 |
| CAD-UL | 102 | 90 |
| CAD-UL | 99 | 68 |
| CARDtools | 20 | 11 |
| CaseTools | 11 | * |
| Ceibo | 41 | 21 |
| CellNet Data Systems | 96 | |
| ChipTools | 101 | 81 |
| CMX Company | 101 | 79 |
| Concurrent Sciences Inc. | 44 | 23 |
| Connect Tech Inc. | 102 | 92 |
| Cosmic Software | 107 | 57 |
| Cypress Semiconductor | 37 | * |
| Datalight | 100 | 74 |
| Dataman Programmers Inc. | 102 | 85 |
| Diab Data | 83 | 49 |
| EBS | 53 | 30 |
| Electronic Engineering Tools | 100 | 70 |
| EMAC Inc. | 90 | 55 |
| Embedded Systems Conf. West | 84 | * |
| Embedded.Com | 89 | * |
| Embedded Systems CD-ROM | 95 | * |
| Embedded System Products | 90 | 14 |
| Embedtec | 103 | 98 |
| Empress Software | 46 | 25 |
| EPI | 103 | 93 |
| Epilogue Technology Corp. | 57 | 34 |
| EST Corp. | 1 | 2 |
| Express Logic | 71 | 41 |
| General Software | 76 | 44 |
| Go DSP Corp | 10 | 7 |
| Grammar Engine Inc. | 50 | 27 |
| Green Hills Software Inc. | 78 | 46 |
| Hitachi America, Ltd. | 21 | * |
| HITECH Equipment Corp. | 56 | 33 |
| Hitex | 47 | 26 |
| Huntsville Microsystems Inc. | C4 | 61 |
| I Logix | 23 | 13 |
| IAR Systems | 75 | 43 |
| Integrated Systems Inc. | 8 | 5 |
| KADAK Products | 10 | 6 |
| Keil Software Inc. | 6 | 4 |
| Kontron Electronik | 22 | 12 |
| Logical Devices | 103 | 95 |
| Lynx Real Time Systems | 69 | 40 |
| Micro Computer Specialists Inc. | 68 | 39 |
| Micro Digital Inc. | 85 | 51 |
| Micro/Sys Inc. | 90 | 56 |
| Microsystems Development | 100 | 76 |
| Microsystems Development | 99 | 62 |
| Microtec | 81 | 48 |
| Microtek International Inc. | 101 | 77 |
| Microtek International Inc. | 103 | 94 |
| Microtek International Inc. | 111 | 59 |
| Miller Freeman Directories | 87 | * |
| Motorola | 25 | * |
| Motorola | 4-5 | * |
| Needham's Electronics | 101 | 82 |
| Nohau Corp. | C3 | 60 |
| Nohau Corp. | 45 | 24 |
| Nohau Corp. | 99 | 65 |
| Noral Micrologics | 100 | 75 |
| Octagon Systems | 101 | 78 |
| On Time | 90 | |
| Pacific Softworks | 77 | 45 |
| Paradigm Systems | 66 | |
| Parallax Inc. | 101 | 84 |
| Phar Lap Software Inc. | 55 | 32 |
| Philips Semiconductor | 15 | 9 |
| Precise Software | 52 | 29 |
| Premia | 59 | 35 |
| QNX | 31 | 16 |
| QNX | 67 | 38 |
| Realtime Control Inc. | 103 | 99 |
| Realtime Innovations | 51 | 28 |
| Scanlon | 101 | 83 |
| Signum Systems | 99 | 63 |
| Softchip | 85 | 50 |
| Softools Inc. | 100 | 73 |
| Software Development Systems | 2 | 3 |
| Spectrum Digital | 86 | 53 |
| SSV | 99 | 64 |
| Tasking | 33 | 18 |
| Tasking | 73 | 42 |
| TechTools | 102 | 86 |
| TechTools | 86 | 102 |
| Tern Inc. | 100 | 72 |
| Texas Instruments | 27 | * |
| Texas Instruments | 29 | 15 |
| TRW | 98 | |
| US Software Corp | 63 | 36 |
| V Automation | 54 | 31 |
| Xeltek | 99 | 67 |
| Z World Engineering | 99 | 69 |

*The index on this page is provided as a service to readers. The publisher does not assume any liability for errors or omissions.*

* These advertisers prefer to be contacted directly.

*by Jack G. Ganssle*

# It's Done

A manager likes nothing more than to hear "It's done." Unhappily, in the embedded world, "It's done" may be about as honest as "Sure, this is an alpha version, but it's ready for the market." The phrase simply means something different to everyone. The ultra-smart engineer might mean he has a pretty good idea about how to create the product. A firmware builder generally means the code has been completely written—more or less. (Tested? Hey, that's just a detail!)

At the risk of outraging you, gentle reader, I suggest that we should look at "It's done" from an absolutist's perspective. "It's done" means it's ready to ship to the customer. Period. If you manage an engineering group, you're being evaluated on your ability to get quality products to market. As a developer, you have to understand and work toward your boss's or boss's boss's needs. Anything less and you're a meaningless cog in a big wheel, someone all too easy to dispense with during the next downturn. Our goal is not to create code or to build widgets—it's getting a product to market. "It's done" means the product is ready to go.

To say "It's done, except for that last bit of optimization" is nothing more than a lie, as that last little task is probably one of a dozen last little tasks, like testing the optimization, or updating the documentation, or going through the release procedure. Yet we know our supervisors want to hear "It's done"—sometimes, when a project is running months late, it's all they dream of—so we do a bit of "truth management" to get them off our back, to make them happier, or to relieve some of the pressure on us. Anything less than total objective honesty to those in charge is the act of a teenager who trades getting in trouble today for much bigger problems tomorrow.

**Maybe I'm a pessimist, but the one thing we can rely on is the perversity of nature: something will go wrong.**

We're all guilty of this dishonesty. I hear it everyday from developers who convince themselves that they are so close to completion that they're all but done, or who feel that one more night of heroic efforts will prove them close enough to being correct. We computer people are the worst sort of egotistical intellectuals. We believe—no, we know—that we're so good that we'll bail ourselves out of the disaster of the hour with a few all-nighters.

Experience proves otherwise. Projects become late, panic mode starts, and sure enough, the system is eventually delivered. All of our promises and plans are for naught, though, as a thousand technical, managerial, and business problems thwart our best efforts. Maybe I'm a pessimist, but it seems the one thing we can rely on is the perversity of nature: something always goes wrong.

A wise boss, on hearing "It's done," will question you extensively to understand the exact level of "done." A novice—or one working in the desperation of excessive hope—may believe "done" means "ready to ship," and start to make commitments that no one will live up to.

## DON'T DO BETTER

Breaking this dysfunctional cycle is up to each of us. But simply promising ourselves to "just do better" is stupid and ineffective.

With age (but less-than-anticipated maturity), it's interesting to look back and see how most of us form personalities early in life, personalities with strengths and weaknesses that stay largely intact over the course of decades. The embedded community is composed mostly of smart, well-educated people, many of whom believe in some sort of personal improvement. But are we really successful? How many of us live up to our New Year's resolutions?

Browse any bookstore. The shelves groan under the weight of self-help books. How many people are actually helped, or at least helped to the point of solving a particular problem? Go to the diet section—I think more diets are being sold than the sum total of excess pounds in the U.S. People buy these books with the best of intentions, yet many fail to achieve their goals.

Our desires and plans for self improvement—at home or at the office—are of the more noble human characteristics, but in reality, we fail—a lot. We commonly compensate by promising to ourselves to "try harder" or to "do better." That approach is rarely effective.

Change works best when we change the way we do things. Forget the vague promises and invent a new way of accomplishing your goal. Plan to reduce your drinking? Exercise regularly? Develop a process that ensures you're meeting your goal. Don't just try harder, but change something basic and log the results daily.

The same goes for improving your abilities as a developer. Forget the vague promises to read more books, or whatever. Invent a solution that has a

better chance of succeeding. Even better, steal a solution that works from someone else. The moral is to accept that we humans do a poor job of living up to our promises, and therefore we need to make grand, sweeping changes in order to improve.

Check out *Quality is Personal: A Foundation for Total Quality Management* (Harry Roberts, Free Press, New York, 1993), a great book about finding new ways to do better. It turns the quality movement inside out, acknowledging our limitations, yet providing a framework for improvement despite human frailties.

Unfortunately, in many circles the term "quality" has been Dilbertized to the point of total cynicism. So many fads, in management and elsewhere, are rammed down our throats that it's understandable when we cringe at the word "quality." Try to swallow your bile and read the book for useful concepts. After all, most of us are indeed seeking different sorts of quality—quality time with the family, quality work at the office that we can be proud of, and even a "quality" inner peace and contentment. I do think it's important to proactively work at improving every area of our lives, and I managed to pull some neat ideas about this from the book.

Applying this concept to overcoming the "It's done" weakness, I think the solution is to develop a policy—and more importantly, a system—of total exposure. We need to recognize that "It's done" means different things to different people.

Nothing beats a dynamic action list. It matters little if you use the latest project management software or a ruled paper pad; what's important is maintaining a list of things that must be done before you can utter those profound and soaked-in-meaning words "it's done" with firm resolve.

No list is effective unless it's updated in real time as the project changes or as you learn more. Planning is always incomplete, a continual source of acrimony as the scope of a project changes. No matter—add the changes to the list, check off items as they're completed, and use the list as your scorecard, showing exactly which parts of a project are done and what remains. Expose it constantly to the entire project team, so everyone knows the project's status. Undisciplined project teams always drop the management tools—like the action list—as soon as they get too busy to keep these tools up-to-date. Ignore your management tools and disaster is inevitable.

## THE WRAP-UP

Your action list must contain everything that is part of the project. End-of-project details are just as important as getting the code out, yet these details are often neglected. Rather than engaging in a mad scramble to clean up the mess in the microseconds between ending one project and starting the next, we can instead invent new approaches that leave no mess to clean up.

Take documentation, for example. Somebody has to write that user manual before the product is ready for the customer. If all of the code is complete and the hardware works, but there's no manual, well, for all intents and purposes, the product doesn't exist. You can't ship it, so the development is still in that "spending money, unable to create revenue" status that so endears us to upper management.

Granted, some documentation cannot be created until the product itself is ready to go, but much can—and should be—developed in parallel with the normal engineering. In fact, I think it makes sense to rethink much of this process. Often a user manual is a much better project specification than the raw specs themselves. Doesn't it make sense, then, to write the manual first, before creating a line of code? Even better, the non-techie MBA types will get a better look at the final product through the customers' eyes when they read the manual, hopefully demanding changes before we start writing code, instead of a week before shipment.

Another often-neglected area is version control. Some outfits defer the issue of controlling software until a release is imminent. Again, this is inefficient and silly. Before starting a project, implement a version control system (VCS). Once this was a difficult process; now it's almost trivial.

Even a one-person shop needs a formal VCS. It's truly magical to be able to rebuild any version of a set of firmware, even a version that is years old. The VCS provides a sure way to answer those questions that pepper every bug discussion, like "when did this bug pop up?" A number of vendors provide these VCSs. Most of my experience has been with Microsoft's Visual Source Safe, a Windows-hosted product that looks a lot like File Manager.

A VCS insulates your source code from all of the developers. Under Microsoft's product, the code sort of disappears, rolled into a database of some sort. Scary stuff! You can get at the code only from the user interface provided by the tool, and cannot change code unless you have the rights to do so. It controls the number of people working on modules, and provides mechanisms to create a single correct module from one that has been (in error) simultaneously modified by two or more people.

Sure, you can cheat the controls on any VCS, as you can cheat your project activities list, but doing so is counterproductive. Maybe you'll save a few minutes of time up front, inevitably followed by hours or days of extra time paying for taking the shortcut.

So save that time by never bypassing the VCS. Check modules in and out as needed. Don't hoard checked-out modules "in case you need them." Use the system as intended, daily, so there's no VCS cleanup required at the project's end.

The same goes for project backups. Develop a system before you start that insures everything is backed up in a reasonable manner at frequent intervals. Don't plan on an end-of-project backup and file cleanup.

In my experience, one can't rely on individuals to be religious about back-

ups. Some are passionately concerned and will do whatever is needed to have every file stored in more than one place. Others are concerned but inconsistent in their efforts. I admit to being anal-retentive about backups. A fire that destroys all of the equipment would be an incredible headache; one that took the data would destroy the business.

Yet, preaching about data duplication and making draconian rules is ineffective. There's a bell curve of responses from people—some will save their files every night, while others forget or get distracted. Clearly, we need an approach that doesn't rely on people's good intentions.

We keep all critical data (including the VCS database) on one server, and run a series of backups of that machine. All backups are automated, and other than changing the tape, require no human intervention. Tapes are great for archival storage, but are somewhat of a pain to use. When something goes wrong, like the person who changes the tapes goes on vacation, or a tape fails, or any of a hundred other problems occurs, data could be lost. To combat this possibility, we keep a spare disk drive in the server just for daily and weekly backups. An automated job starts every night and copies all critical data to the spare disk. On Saturdays another copy is made to another hard disk, as added insurance.

Extreme measures? Maybe. We don't lose data, though—ever. Computers fail and hard disks crash, but (to date) the data stays intact. And in the spirit of inventing a system that runs well despite human weaknesses, the process is automatic and unattended.

With a VCS, though, people do have the ability to check out files and leave them out for long periods of time. Conceivably, their work on these files could be lost in a system crash. Short of rolling all network drives to tape (difficult, as most developers try to limit access to their shared drives), the solution we use is simply to have the VCS police monitor checked-out files. The project manager checks the status of files under his or her control (the VCS tells which files are checked out to which developers), and ensures that nothing stays out too long. I'm still looking for a better solution to this potential problem.

## STANDARDS

The quintessential software guru's observation, "Well, the code is working, now I just have to comment it" signals the death of schedules and product quality. Leaving any part of the development process until the end will further fuzz the meaning of "It's done."

Software that isn't created to a uniform and rational standard cannot succeed over the long term. Software just hacked together and shipped as soon as it seems to work is an open sore that will require maintenance, more than you ever expect, and possibly forever.

The solution is to employ a software standard that is rigorously enforced. I've written about this solution before in *ESP*, and I received lots of correspondence from people looking for a standard they could use. One such standard can be found at http://www.microsol.com. I'm sure there are lots of others. If you have a standard to share with the rest of us, please let me know and I'll publish the URL here.

Be honest with yourself and create decent code you're proud of, create reasonable action lists or project schedules, and always search for new ways to solve old problems. Make the phrase "It's done" mean something profound. ESP

*Jack G. Ganssle is an old (well, not so old) embedded hand. He helps companies improve their engineering teams and analyzes embedded product designs. Ganssle enjoys contact—reach him at jack@ganssle.com.*

# Byte by Byte

Both C and C++ are designed to be machine-independent languages that nevertheless remain reasonably close to the underlying machine architecture. To be portable, they must define clearly and precisely an abstract architecture that can adapt to a variety of real world computer architectures. Nowhere are clarity and precision more important than for the fundamental model of how to represent scalar data. To know what constitutes a proper `int` or `double` object, you must certainly know what's in a byte.

Embedded programs in particular often deal in that netherworld between raw storage and a properly up-and-running "C machine." Embedded programs often run on hardware that strains the bounds of compliance with programming language standards. It's important to know what you can and cannot get away with, and still enforce the basic guarantees required by C and C++.

Over the years, the C and C++ standards committees have sharpened their collective understanding of the storage model shared by the two languages. My goal here is to summarize that abstract model, including some aspects that have only recently become clear.

C was born on the DEC PDP-11. In short order, it was moved to a variety of other computer architectures. Most of those early machines supported byte-level addressing of 8-bit bytes. They performed integer arithmetic in two's complement with no overflow trapping. The primary difference among those early machines was in the order of bytes within multibyte scalar objects. Needless to say, this commonality of basic features materially eased the problems of porting C among machines.

Nevertheless, C soon appeared on machines that didn't follow this agreeable recipe. Some machines had word-oriented architectures, where the further subdivision into bytes was both tiresome and arbitrary. Eight-bit bytes were not always the obvious choice, particularly for a machine with, say, 36-bit words. Some machines were less tolerant about integer arithmetic. They would poorly support C's brand of unsigned arithmetic, or they would have codes for -0 that sometimes changed to +0, and sometimes did not. For these more diverse architectures, knowing how best to implement C was difficult.

**When the ANSI standardization effort began for C, few architectures existed that didn't support some flavor of C.**

By the time the ANSI standardization effort began for C in 1983, few architectures existed that did *not* support some flavor of C. A diversity of implementations had arisen, some inevitably mapping C to the underlying hardware in different ways. One of the important early questions for the C Standard was whether any of these existing implementations had stretched the unwritten rules too much. Put another way, the committee had to determine the invariant properties of an abstract "C machine" across diverse computer architectures.

Many years have passed since that first round of questions were answered. The decisions made then by committee X3J11 (now just J11) have held up rather well, though they have enjoyed occasional refinement. Here is a brief summary of the C memory model, vintage 1984.

First, it was clear that type `char` was in some sense elemental. A common idiom for copying an arbitrary object of type `T` from `x` to `y` was:

```
char *p = (char *)&x, *q = (char *)&y;
char *pend = p + sizeof (T);
while (p < pend)
  *p++ = *q++;
```

Modulo various optimizations, that idiom is still widely used in C programs. Committee X3J11 essentially took the correctness of this idiom as axiomatic and worked backwards. What must be true for the idiom always to work properly? Each line of the idiom expresses a different constraint:

- The first line says that a pointer to an arbitrary object can be converted to a pointer to `char` without loss of information
- The second line says that an object of type `T` is exactly as large as an array of `sizeof (T)` elements of type `char`
- The third line says that it is permissible to compare a pointer within an array to a pointer just beyond the end of the array
- The fourth line says that copying the value of each `char` in the (conceptually) overlapping array suffices to copy the value of the object of type `T`

Note that these constraints apply both to the addressability of objects (constraints 1 and 3) and their representation (constraints 2 and 4). Earlier programming languages were less ambitious in defining the effects of

pointer arithmetic and aliasing (treating the same storage as two or more different types of objects). Hence, C had to invent semantics that it couldn't inherit from its predecessors.

The copy idiom has two other important guises. The idiom is implicitly buried in such library functions as `memcpy` and `memmove`, declared in `<string.h>`. It is also implicitly at work when reading and writing binary files, such as with `fread` and `fwrite`, declared in `<stdio.h>`. In all cases, it is fundamental to C that the value of an object can be copied—even out to a file and back—by treating it as a sequence of objects of type `char`.

There's more to a data representation than copying bits about. An implementation has to assign values to the different bit patterns. It also needs to provide some assurance, for writing portable code, that a given object type supports an adequate range of values as well. So X3J11 also found itself in the business of decreeing constraints on representations and minimum ranges for the scalar types of C. You can find many of these decisions memorialized in the headers `<float.h>` (invented by the committee) and `<limits.h>` (borrowed from Posix).

An unsigned integer is the easiest to describe. It must be represented as a weighted binary code. In other words, $N$ bits always represents the inclusive range of values zero (all zero bits) through $2^N - 1$ (all one bits). Moreover, the committee set the following minimum values of $N$ for the unsigned integer types:

| $N$ | Type |
|---|---|
| 8 | unsigned char |
| 16 | unsigned short |
| 16 | unsigned int |
| 32 | unsigned long |

A signed integer has a little more latitude. It, too, must be represented as a weighted binary code, but with a difference. One of the $N$ bits serves as a sign bit. If that bit is zero, the remaining $N - 1$ bits represent positive values by the same rules as for unsigned integers. If the sign bit is set, however, several encodings are permissible. All must have distinct codes for the inclusive range of values $-(2^N - 1)$ through $-1$. The remaining code happens to represent:

- $-(2^N)$ in two's complement encoding
- $-0$ in one's complement
- $-0$ in signed magnitude

All are permissible encodings for Standard C.

Floating-point arithmetic is even more tolerant. Within broad limits, Standard C pretty much tolerates whatever the underlying architecture does in the way of floating-point arithmetic. This is such a huge topic that I won't begin to probe its depths here. It warrants at least another essay of its own.

The same goes for pointers. All I'll say here is that Standard C tolerates a number of different representations for pointers, even within a single implementation. One value of each representation must represent a null pointer—designating no object that you can declare or allocate within the program. But a null pointer need not be all zero bits. It might be a good idea, for other reasons, to make all zero bits stand for a null pointer, but Standard C doesn't require this nicety. (And yes, floating-point representations have the same latitude. Having all zero bits doesn't necessarily represent a floating-point zero.)

## CHALLENGES

What I've just described is essentially what was captured in the C Standard. It wasn't long, however, before people began to poke at the corners of the tidy little model set out above. Computer architectures are endlessly diverse, it seems. And sooner or later someone will want to put up something resembling C on practically every piece of hardware that can execute a stored program. Here are some of the issues that have caused a few heated discussions among those who would interpret the C Standard.

Perhaps the most serious is the very notion of "byte" itself. The C Standard is fairly glib about confusing three distinct concepts:

- a byte, the elementary unit of storage
- type `char`, the smallest of the scalar types
- a character, which is a code value that stands for a member of some character set

Within a C program, the murkiness isn't too troublesome. A `char` object does indeed occupy one byte. That's because the `sizeof` operator officially counts bytes, and `sizeof (char)` always has a value of one. And a `char` object can, by definition, store the codes for all the characters in the basic C character set, and then some.

Of course, it doesn't help that even the C Standard has to talk about multibyte characters and wide characters as well as (single-byte?) characters. Both the C Standard itself and the more recently approved Amendment 1 have rightly been accused of fuzziness in distinguishing all the various flavors of characters now kicking about. But that's not the issue here.

The real problem is that the outside world has settled on a fairly sharp definition of a byte. It's what ISO standards tend to call an "octet," or group of eight bits. The world, as we all know, is paved with 8-bit UARTs, communication channels, and magnetic media. An implementation that can't traffic in octets is doomed to life in a tiny ghetto. Indeed, such was the fate of the ambitious C Machine, sold for a time by Bolt, Beraneck, and Newman. That machine's oversize bytes caused no end of grief.

Other machines are still viable despite having oversize bytes. A word-

oriented processor, for example, might want to avoid the inefficiencies of byte packing and unpacking by defining a byte as big as a word. Standard C does indeed permit:

```
sizeof (char)
 == sizeof (short)
 == sizeof (int)
 == sizeof (long)
 == 1
```

so long as the minimum size rules I gave above hold true. That's all well and good, but what do you do about reading and writing more conventional files?

You can indeed make a conforming C implementation with these constraints, but at a cost. Text files have sufficient latitude to permit reading and writing of octets, regardless of internal byte size. Binary files have to deal exclusively in words—commandeering a sufficient number of octets to represent each word exactly. If a word is not a multiple of eight bits, then you lose a few bits on the way in and add padding on the way out. This isn't a transparent channel to another implementation, but it can serve as a transparent repository for objects in local programs.

The real problem is more pragmatic. Like it or not, the world is full of C code written with additional constraints in mind. Many programs misbehave unless:

```
sizeof (char) < sizeof (int)
```

regardless of what the C Standard says. Many programs assume that a byte is indeed an octet. The cachet of official conformance may be useful for an unconventional implementation, but that doesn't guarantee easy access to supposedly portable C code.

More interesting are implementations that insist on leaving "holes" in integers. Consider an architecture with 36-bit words and only signed integer operations. The best performance might be obtained by representing `unsigned long` with 32 to 35 magnitude bits, keeping clear the sign and any other unused bits. Does this practice conform to the C Standard?

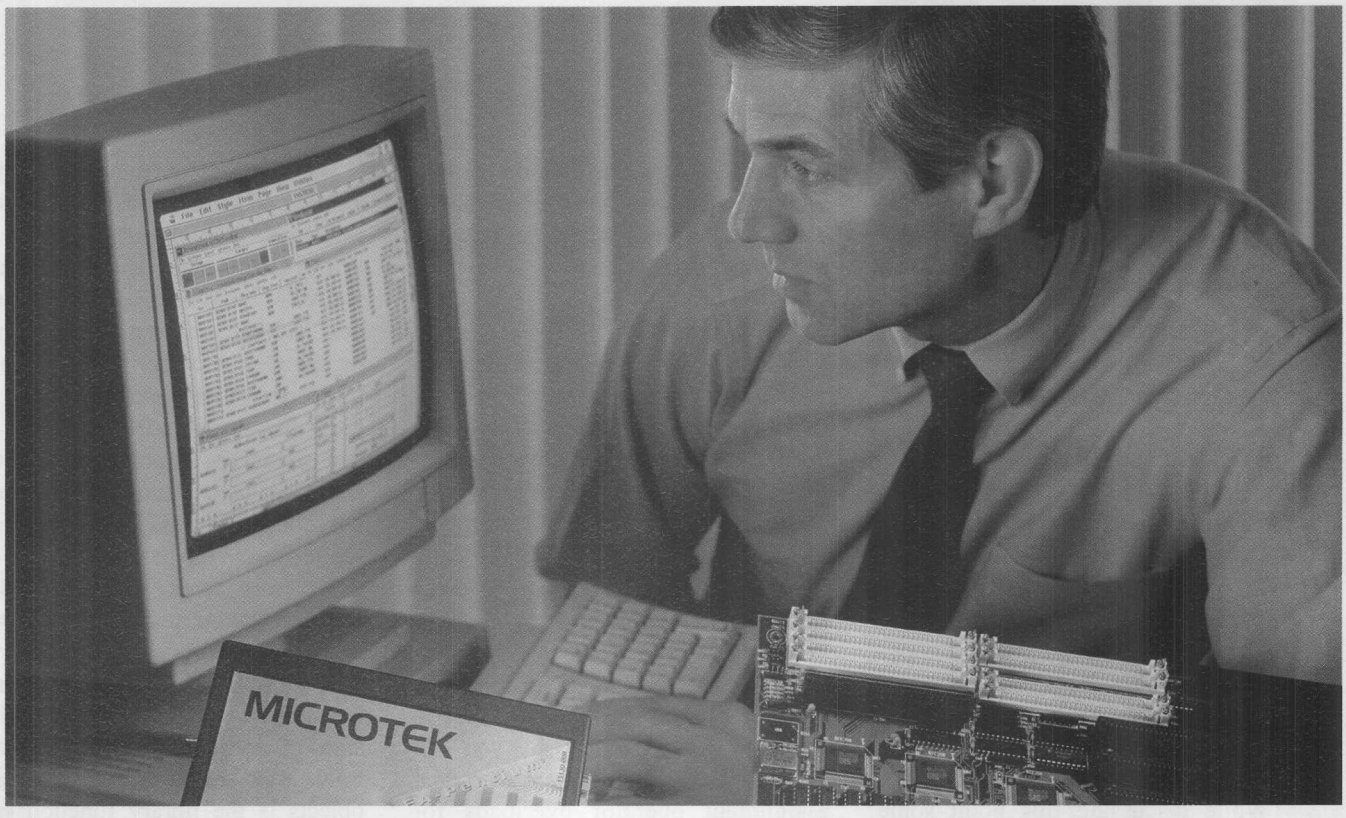
If you buy that, then consider an architecture that has only floating-point arithmetic. The best performance here might be obtained by representing `unsigned long` with 32 or more mantissa bits and an exponent that ensures that the mantissa looks like an integer. Does this conform?

I've even been asked about an architecture that leaves explicit holes *inside* an integer. Bit 23, say, may be weighted $2^{23}$ and Bit 30 may be weighted $2^{24}$. Bits in between are weighted zero. Whatever you may think of the aesthetics (or ancestry) of the architect, such machines have been made and have attained some degree of commercial importance. Does this bizarre practice still conform to the C Standard?

It turns out that the answer in all three cases is yes. The C Standard failed to make it clear, but an implementation can indeed distinguish between the *object* bits that make up a representation and the *value* bits that contribute to its stored value. It's okay to have object bits that aren't value bits, at least in many cases. They don't even have to be in any particular order, as long as the arithmetic comes out right.

This latitude is a bit more constrained for type `char`, however. This type can't have any unused object bits in places that would overlap value bits in another object type. Otherwise, the copy idiom we discussed would fail. Note that saying that type `char` can have no unused object bits is too strong a statement. Many memories have a parity bit on each byte, which C tolerates quite nicely.

A still more interesting issue has been recently raised. Type `char` can have the same representation as either `unsigned char` or `signed char`. In the latter case, several encoding rules are acceptable, as previously described.




# "If I could find it, I could fix it."

SEE your code at work. Find timing conflicts fast that are invisible with logic analyzers and software debuggers.

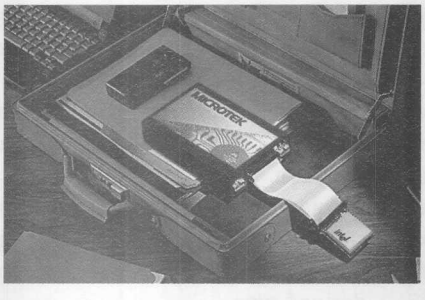
Microtek In-Circuit Emulators combine a state-of-the-art source level debugger with the most advanced event trigger and trace system available.

**160-bits Wide by 256k Trace with Clock-edge Resolution.**
With this much trace and smart triggering, you can record virtually every event. And events can be followed right back to their source code without stopping the target.

**SWAT™ Software Analysis Tool**

- Fast, easy code validation for design engineers.
- Built into Microtek In-Circuit Emulators.
- Code Coverage and Performance Analysis offer you design spontaneity without instrumenting your code!

CODE CHANGES




**Smaller is Better.**
Compare today's Microtek In-Circuit Emulators that fit in your briefcase, with the traditional "chassis" of just two years ago. The difference is remarkable! The PowerPack®EA for the Pentium® processor is only 7.2" x 4.6". And the probe tip is smaller than your business card, so it fits into the tightest targets.

**Call for FREE ICE Tips AppNotes:**

**1 (800) 886-7333**

www.microtekintl.com
Phone: (503) 645-7333
Fax: (503) 629-8460

# MICROTEK
IN-CIRCUIT EMULATORS

NEW PRODUCTS
Three Emulators for Pentium® Processors
SWAT™ Software Analysis Tool
Two National NS486™ Emulators
High-Performance 80C186 Emulator

**Microtek In-Circuit Emulators for the following processors:**

Pentium®• Intel486™• National NS486™• Intel386™EX • 386DX • 386CX/SX • 80C186 • 8051
68360 • 68340 • 68F333 • 68332 • 68331 • 68330 • 68HC16 • 68328 • ColdFire

**CIRCLE # 59 ON READER SERVICE CARD**

Two of them, ones complement and signed magnitude, have a representation for -0. So what happens if the venerable copy idiom runs on such an implementation, and the hardware likes to convert -0 to +0 at the drop of a hat? From all appearances, the loop no longer transparently copies object bits.

Evidently, appearances are correct, at least in principle. Few, if any, machines actually represent type `char` this way. Those machines that do may well copy bytes transparently anyway. Nevertheless, it is mildly rash for the C Standard to talk so glibly about copying arrays of `char`. Instead, the standard should emphasize that arrays of `unsigned char` are the proper aliases to use for copying objects in C.

While discussing this point some time ago, X3J11 considered still another related issue. I stated earlier that signed integers have one unencumbered bit pattern. Sometimes it stands for a large negative number, sometimes for -0. But are those the only choices?

In particular, floating-point representations can have a host of NaN (Not a Number) codes. Some of these codes can even trigger a hardware trap when accessed from an object (these are "signaling NaNs," in contrast to "quiet NaNs"). Similarly, some machines can detect and trap on invalid pointer values. Should integers be exempt from this treatment or are they fair game too? A system with aggressive debugging support might want to add such a capability, while a program that plays with uninitialized integers might not want to. In the interest of maximum portability, everyone should know what is permitted.

You can argue it either way, and we did. In the end, however, the sentiment among X3J11 members was that signed integers can indeed have a NaN code. Note that this license is not granted unsigned integers. All value bits participate in the value. Of course, a debugging compiler might still choose to tag uninitialized unsigned integers in some way. And some users might still welcome a report that an uninitialized unsigned integer was accessed. Conformance is often of secondary importance when you're chasing a nasty bug.

## C++ CONSTRAINTS

I shall end with a few comments on additional issues raised by C++. Perhaps the most important issue is that the copy idiom isn't always appropriate. Some objects in C++ are indeed containers for values. Make a copy of their bytes and you have a valid copy of their value. But other objects don't suffer bitwise copying as well. They may store reference counts or pointers to other parts of the same object. In such cases, copying is best left to smarter agents. That's why you can define specific copy constructors and assignment operator overloads for classes in C++.

An object can also have a dynamically determined type. Put simply, you can have a pointer to an object and not know whether it is truly an object of the specified base type or of some derived type you don't even know about. In practical terms, each such object must store a pointer to a table of information. That table is usually called a "virtual table," or v-table, because its principal function is to store an array of pointers to the virtual functions that might be overridden in a derived type.

In more practical terms, such an object requires a bit of initializing, even before its constructor is called. Otherwise the programmer would be saddled with all sorts of caveats about taking addresses and calling virtual functions while the constructor executes. And that situation calls for further refinements of the memory model. Some distinguishable states are:

- The address of the object is representable
- The object is ready to be constructed
- The object has been successfully constructed
- The object has been destroyed
- The address of the object is no longer representable

By contrast, Standard C seldom worries about more than the first and last states of an object.

The C++ Standards committees, J16 and WG21, have refined these issues to deal with the greater precision required by the C++ memory model. The distinctions are important because it is easy for programmer-supplied code that can access an object in all of these states to be executed. Unfortunately, not all the ground rules have been spelled out in the past. Some C++ code doubtlessly works only because of implementation peculiarities, not for any sound theoretical reasons.

A particularly delicate area is the business of programmer-supplied versions of `operator new`. Consider that the only real source of unconstructed objects is the library function `malloc` and its ilk, all declared in `<stdlib.h>`. Any other storage in a C++ program has already had some object constructed on it. (See "Turtles and Overcoats," *ESP*, April 1997, p. 100.)

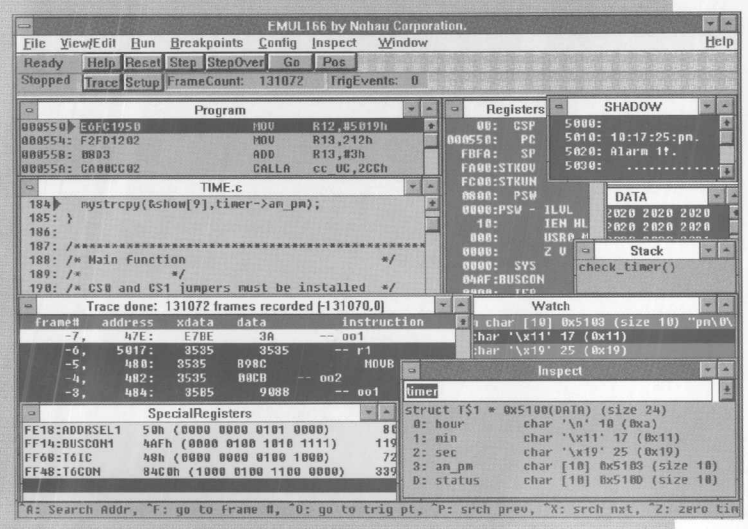
Can you destroy the old object, then construct the new one in the same space? Or can you simply construct the new object, knowing that the old object needs no destroying? Can you double construct an object safely? Serious C++ code indulges in all these practices, but no guidelines have yet been laid down to say whether any or all are safe.

As a result, I venture to say that the memory model that began with C will continue to be refined. We know a lot more about how to structure storage than your basic assembly language jockey, but we also have more to learn. At stake is the safety of optimizations and the ultimate portability of large quantities of code, in both C and C++. This essay is merely an interim report on a work in progress. ESP

*P.J. Plauger is President of Dinkumware, Ltd. and author of the standard C++ library shipped with Microsoft Visual C++. His latest books are* The Draft Standard C++ Library *and* Programming on Purpose *(three volumes), both published by Prentice Hall in Englewood Cliffs, NJ.*

# nohau

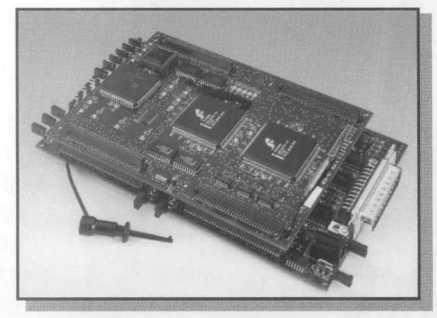
# C166 In-Circuit Emulators




## Real-Time Microprocessor Development Systems

## EMUL166™-PC

*To learn more, call* ***(408) 866-1820*** *for a product brochure and a FREE Demo Disk. The Demo can also be downloaded from our web site-* ***http://www.nohau.com***



*For more information via your Fax, call our 24-hour Fax Center at (408) 378-2912.*

### EMUL166-PC FEATURES:

- Supports Siemens' C166, C167CR, C167SR, C165, C163, C161 and SGS Thomson ST10 series.
- Hosted on PC's or workstations.
- High-level support for Keil and Tasking C-compilers.
- Connects to PC through standard printer port or through plug-in board.
- Optional trace board.

MICROSOFT WINDOWS/'95 COMPATIBLE


nohau CORPORATION

51 E. Campbell Avenue
Campbell, CA 95008-2053
Fax. (408) 378-7869
**Tel. (408) 866-1820**
Email: sales@nohau.com
*web: http://www.nohau.com*


**Support also available for:**

**8051 P51XA™ MCS®251 80C196 MCS®296 68HC11 68HC12 68HC16 683xx**


Argentina 54 1 312-1079/9103 , Australia (02) 654 1873, Austria 0222 277 20-0, Benelux (078) 681 61 33
Brazil (011) 453-5588 , Canada 514-689-5889, China +86-10-2059495, Denmark 43 44 60 10
Finland 90-887 33 330 , France (1) 69 41 28 01, Germany 07043/40247 , Great Britain 01962-733 140
Greece +30-1-924 20 72, India (0212) 412164, Israel 03-6491202, Italy 02-49-82-051, Japan (03) 3405-0511

NOHAU INTERNATIONAL

Korea (02) 784-7841, New Zealand 09-3092464, Norway 22 67 40 20, Portugal 01 421 3141
Romania 056 200057, Singapore +65 749-0870, S.Africa (021) 234 943, Spain (93) 291 76 33
Sweden 040-59 22 00, Switzerland 01-745 18 18, Taiwan 02 7640215, Thailand (02) 668-5080


**CIRCLE # 60 ON READER SERVICE CARD**

# HMI. Putting the power behind PowerPC development.


HMI's SPS-2000 Series PowerPC development tools take emulation technology to a whole new level. True in-circuit emulators, they change easily and economically to support most PowerPC variants from Motorola and IBM via a simple, low-cost pod change. And they have the following standard performance features that make them the industry's most powerful microprocessor development system: • Hardware-based non-statistical software performance analyzer. • Ethernet interface for high-speed connectivity. • High-speed overlay memory for code download, mappable via chip selects. • Shadow RAM for real-time data variable/memory monitoring. • 128K deep trace buffer configurable in up to 8192 separate buffers for windowing events of interest. • 8-level trigger and breakpoint sequencing logic. • Time-based delays for break and trigger points. • Direct variable editing in watch windows. • Native GUI support from multiple host platforms (Windows 3.1x/95/NT, Sun, HP) using SourceGate II, HMI's powerful source-level debugger that is a common user interface for all HMI products. • Multiple object file format support. Use your compiler of choice! • Unlimited CodeView windows allow breakpoints to be set across multiple modules displayed in source, assembly, or a combination of the two. • Can be operated stand-alone with no target system required or can be put in-circuit in place of the processor.

**HMI ALSO PROVIDES SUPPORT FOR THESE PROCESSORS.**

| | | |
|---|---|---|
| MPC8xx | 68302 Family | 8051 |
| IBM40x | 68306 | 8085 |
| MPC505 | 68307 | 8096 Family |
| ColdFire | 68330 | NSC800 |
| | 68331 | Z80 |
| 68000 | 68332 | 64180/Z180 |
| 68020 | 68333 | 6809 |
| 68030 | 68340 | 68HC11 Family |
| 68040 | 68349 | 68HC16 |
| 68060 | 68360 | 68356 |

• **Free** lifetime technical support (no costly yearly support contracts to worry about—ever!)

Interested? Call or write and we'll be happy to send you more detailed information. Can't wait? Visit our Web Site at http//www.hmi.com/ for instant access to our SPS-2000 data sheet and information on our $199.00 Background Mode Debugger!

HMI™

**HUNTSVILLE MICROSYSTEMS, INC.**

P.O. Box 12415
Huntsville, AL 35815
Tel: (205) 881-6005
Fax: (205) 882-6701
sales@hmi.com
http://www.hmi.com/

MOTOROLA EMBEDDED PLATINUM DEVELOPER

**CIRCLE # 61 ON READER SERVICE CARD**